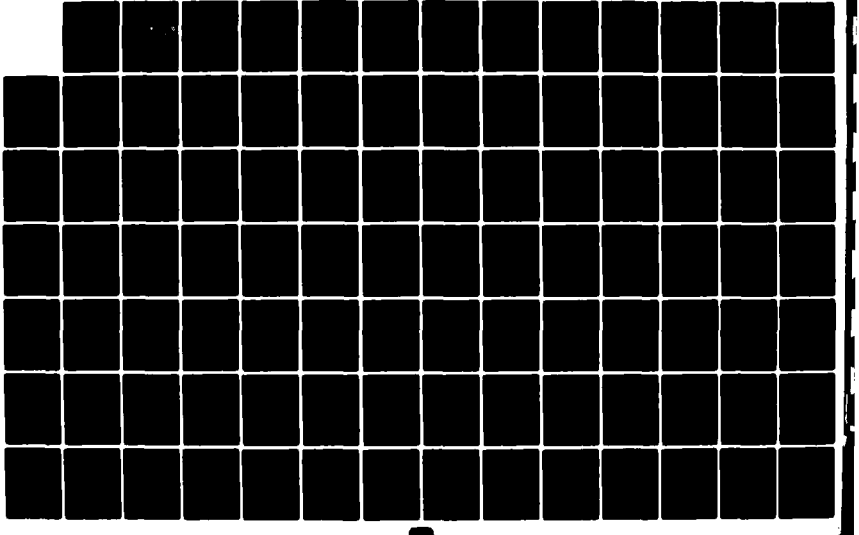
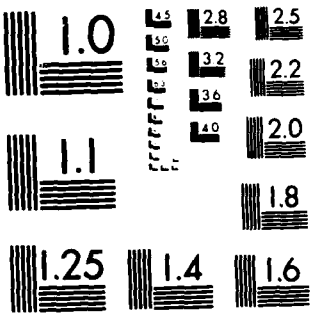
AD-A127 676 A SURVEY OF SOFTWARE QUALITY ASSURANCE METHODS AND AN EVALUATION OF SOFTW..(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M FUQUA ET AL. DEC 82 1/2

UNCLASSIFIED F/G 14/4 NL





NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A SURVEY OF SOFTWARE QUALITY
ASSURANCE METHODS AND AN EVALUATION
OF SOFTWARE QUALITY ASSURANCE
AT FLEET MATERIAL SUPPORT OFFICE

by

Michael Fuqua
Julius Sisco
James Conroy

December 1982

Thesis Advisor: Norman Lyons

Approved for Public Release; Distribution Unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A127 676	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Survey of Software Quality Assurance Methods and an Evaluation of Software Quality Assurance at Fleet Material Support Office		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis, December 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael Fuqua, Julius Sisco, James Conroy		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December, 1982
		13. NUMBER OF PAGES 118
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Quality Assurance, Software, Civilian and Military Software Quality Control, Software, Testing Measurement and Evaluation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

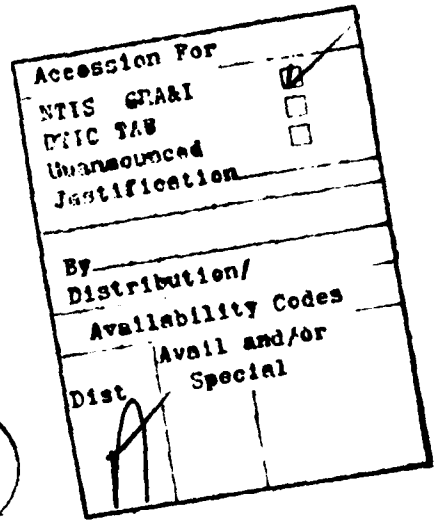
This paper is a survey of existing literature describing software quality assurance and an indepth evaluation of both selected industry quality assurance functions and the Fleet Material Support Office (FMSO) Quality Assurance Division. Quality control at FMSO is effected by the organizational element that produces the product and by a small, centralized staff. Improved systems development and a higher level of quality control are the goals of FMSO. The recommendations and conclusions offered are based (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued) Block 20

on an extensive literature search of existing material on software quality assurance, an indepth study of selected industry quality assurance departments, and an examination of the current state of quality control procedures at FMSO. These recommendations, if implemented, should serve to improve the quality control at FMSO and assist the organization in achieving their goals.



DD Form 1473
1 Jan 73
S/N 0102-014-6601

Approved for public release; distribution unlimited

A Survey of Software Quality Assurance Methods and an Evaluation of Software Quality Assurance at Fleet Material Support Office

by

Michael Fuqua
Lieutenant, U.S. Navy
B.S., Utah State Univ., 1974

Julius Sisco
Lieutenant, U.S. Navy
B.S., Iowa State Univ., 1976

and

James Conroy
Lieutenant Commander, U.S. Navy
B.S., Kearney State College, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
December 1982

Authors: Michael Fuqua Julius Sisco

James Conroy

Approved by: Norman R. Lyons
Thesis Advisor

Dan C. Boger
Second Reader

Chairman, Department of Administrative Sciences

W. M. Woods
Dean of Information and Policy Sciences

ABSTRACT

This paper is a survey of existing literature describing software quality assurance and an indepth evaluation of both selected industry quality assurance functions and the Fleet Material Support Office (FMSO) Quality Assurance Division. Quality control at FMSO is effected by the organizational element that produces the product and by a small, centralized staff. Improved systems development and a higher level of quality control are the goals of FMSO. The recommendations and conclusions offered are based on an extensive literature search of existing material on software quality assurance, an indepth study of selected industry quality assurance departments, and an examination of the current state of quality control procedures at FMSO. These recommendations, if implemented, should serve to improve the quality control at FMSO and assist the organization in achieving their goals.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The computer industry has gradually evolved over the years from the massive hardware systems that were very expensive to build and maintain to the present state of the art where a tiny one quarter inch square chip has more computing power, is enormously cheaper to buy, and can be maintained virtually 100 percent of the time. Along with the hardware change we have seen an even greater improvement in the software development from simple mathematical computation to the launching and recovery of manned space vehicles with greater reliability and performance than at any other time in history.

This evolution has not happened by accident. The development has been based upon making mistakes, documenting those mistakes, and proceeding on. As people continued the process they studied past documentation and continued the documentation process until it has become an accepted part of the development cycle. Although no systematic approach was utilized, there has been an ever increasing tendency towards the development of a set of standards that could provide an avenue for common understanding with a minimum of confusion.

The growth of systems software which occured during the past two decades has presented each organization with continuing challenges in maintaining an effective organizational structure and in following efficient systems development methods and strategies which can be transferred from one entity to another with a maximum degree of cohesiveness and precision and a minimum of ambiguity and confusion.

According to D. Ross of SofTech Inc. [Ref. 1], the quality of software is relative to the intended application and can only be achieved by a disciplined methodology in which quality requirements are initially applied to the original requirements definition for the problem and are then carefully checked and confirmed at every stage in the production process. In order to have any sensible treatment of quality every aspect of the system life cycle must be based on an orderly, controlled, and disciplined methodology. This is not to say that all software must be produced exactly with the same set of tools and techniques, for this clearly would be excessive. There may be a broad spectrum of the degree to which the ideal system technology is approached, but even the simplified, streamlined methodologies must be complete and consistent. Each version must be, in some reasonable fashion, a proper degeneration of the elaborate, most advanced state of the art, merely simplified to suit a simpler set of circumstances. It must be incumbent upon all persons that are engaged in the project to combine company standards and common sense to arrive at the required level that will ensure a quality product.

To accomplish this, there exists within each organization discussed in this paper a group whose primary responsibility is ensuring that company standards are enforced. The Quality Control/ Quality Assurance Branches are the organizations tasked with the job. It is therefore the purpose of this paper to examine the Quality Control programs of various government and non-government organizations that produce software and have established, well documented standards. The effectiveness of their program and the method of operation within their company structure will also be discussed.

In Chapter 2, the authors will list and identify current trends and state of the art processes, techniques, and methods that have been collated through an examination of current literature about quality and the software development process. In Chapter 3 and Chapter 4 we will discuss the Quality Control programs at TRW, General Electric, Naval Oceans Systems Command, and the Fleet Material Support Office. Finally, in Chapter 5, a set of recomendations with justification will be provided for FMSO consideration during the planning and execution of future corporate policy and expansion with emphasis on the Quality Control effort.

II. QUALITY ASSURANCE TECHNIQUES

The rapid expansion of the computer industry in the past ten years has been accompanied by an increase in the problem of producing a quality software product. Surveys have shown that as much as 60 percent of software produced have serious faults in the first iteration - faults serious enough to cause the program to fail in its task [Ref. 2]. A traditional facet of American manufacturing has been strict adherence to quality standards and the production of software should be no different. Although computer software is sometimes considered more of an art than an engineered product, computer professionals agree that software quality assurance is an important part of the software life cycle and most data processing departments now contain some sort of quality assurance function. Many of the significant elements of production engineering such as documentation, testing, project management and quality assurance are now emerging as an integral part of a software production activity. Such things as structured programming, software engineering and quality assurance are fast becoming the norm rather than the exception. User satisfaction, compliance with approved methods of building applications, organizational goals, and performance goals have all been driving forces behind this movement. This chapter deals with quality assurance in the computer industry and its role in software development.

A. QUALITY AND QUALITY ASSURANCE

1. Quality Defined

Quality is, at best, a relative and subjective measurement. The American Heritage Dictionary defines quality as a characteristic or attribute; a property. It also is thought of as the natural or essential character of something. In everyday life, quality measurements are done continually and usually without second thought. Side-by-side comparisons of objects under identical conditions and with predetermined concepts form the basis of most comparitive judgements. Unfortunately, these decisions are usually unique and have little value to anyone else unless they are made by an expert [Ref. 3]. One widespread opinion is that, by its very nature, quality defies definition and must be uniquely defined for the item in question by stating a list of characteristics and attributes. This technique implies no evaluation or judgement of the item, but merely provides descriptive traits by which the appraiser may form an opinion [Ref. 4]. Ken Johnson, a software quality assurance manager in industry and chairman of a working party set up by the Electronic Engineering Association concerned with software quality assurance, disagrees somewhat with other definitions concerning the ephemeral attribute of quality and declares that quality is "the totality of features and characteristics of a product or service which bear on its ability to satisfy a given need. In short, it is a fitness for a purpose at an economic cost" [Ref. 2]. A recent study pointed to a number of myths concerning quality in organizations. These include such things as quality is impossible to measure, quality lowers productivity, quality means poor workers, and quality is the responsibility of the "quality department". It goes on to give the sharpest definition of quality-"quality is the sum costs of prevention,

appraisal and failure" [Ref. 5]. This definition sheds light on the subject but is still not as comprehensive as may be desired. Hence, expectations concerning quality measurements must be tempered with realistic knowledge that any measurement will be partially imperfect or imprecise. When dealing with software, in itself not the most tangible of products, confidence levels and error tolerances play an important role in determining acceptable quality levels.

2. Quality Assurance Defined

Unlike the concept of quality, which is usually thought of as an attribute of a good or service, quality assurance is most often related to a process or methodology. According to Frank Ingressia from TRW Corporation's Defense and Space Systems Group (DSSG), "quality assurance is a lot like sex, freedom and democracy. Everyone is for it, but only under certain conditions." There are a host of definitions which apply to quality assurance. The official Air Force position is that quality assurance is a discipline which provides adequate assurance that material, data, supplies and services conform to established technical requirements and achieve satisfactory results [Ref. 6]. This definition is much more encompassing than most early interpretations which called for quality assurance to merely verify conformance to specifications. At the other end of the spectrum is the feeling that quality assurance is merely the business of ensuring that the product is not the result of good luck but rather the inevitable reward for good management practices. Still another definition and perhaps one that will become widely used has been proposed by the Institue of Electrical and Electronic Engineers in their recent study concerning software quality assurance standards [Ref. 7]. It states that "quality assurance is a planned and systematic pattern of all actions necessary to provide

adequate confidence that the item or product conforms to established technical requirements". This definition has been borrowed from MIL-STD-109B and is consistent with the accepted usage of the term. Conforming to specifications, ensuring good management practices, testing of requirements, confidence that the system is reliable or the product is desirable--all of these things are considered goals of a quality assurance plan. The bottom line for a quality assurance function is ensuring that user's needs have been adequately satisfied. There are a multitude of different philosophies concerning the achievement of these goals and they will be briefly addressed in a later section of this chapter.

3. Traditional Quality Assurance in Industry

For manufactured products, quality usually means a combination of quality of design and of manufacture [Ref. 8]. Quality of design is the value inherent in the design; a measure of the excellence of the design in relation to the customer's requirements. The quality controller has the responsibility to ensure that the quality level determined by management can be achieved on production equipment. Quality of manufacture is a measure of how well the product, at acceptance, conforms to the design. There are most often five basic stages of quality control in a factory [Ref. 8]. These consist of: (1) Deciding what to manufacture and prepare specifications covering all requirements, (2) Make pre-production checks and work out organizational responsibilities, (3) Production, (4) Feedback on quality deficiencies, and (5) Establish long-term quality plans.

The quality control function in a manufacturing plant is usually a very complex and intricate organization. It becomes involved in all phases of the production process

and serves as the ongoing checker for production results. The role, structure and objectives of quality assurance in software production will be examined in detail later in the chapter, but there are many similarities which exist and indeed industrial quality assurance forms the basis for

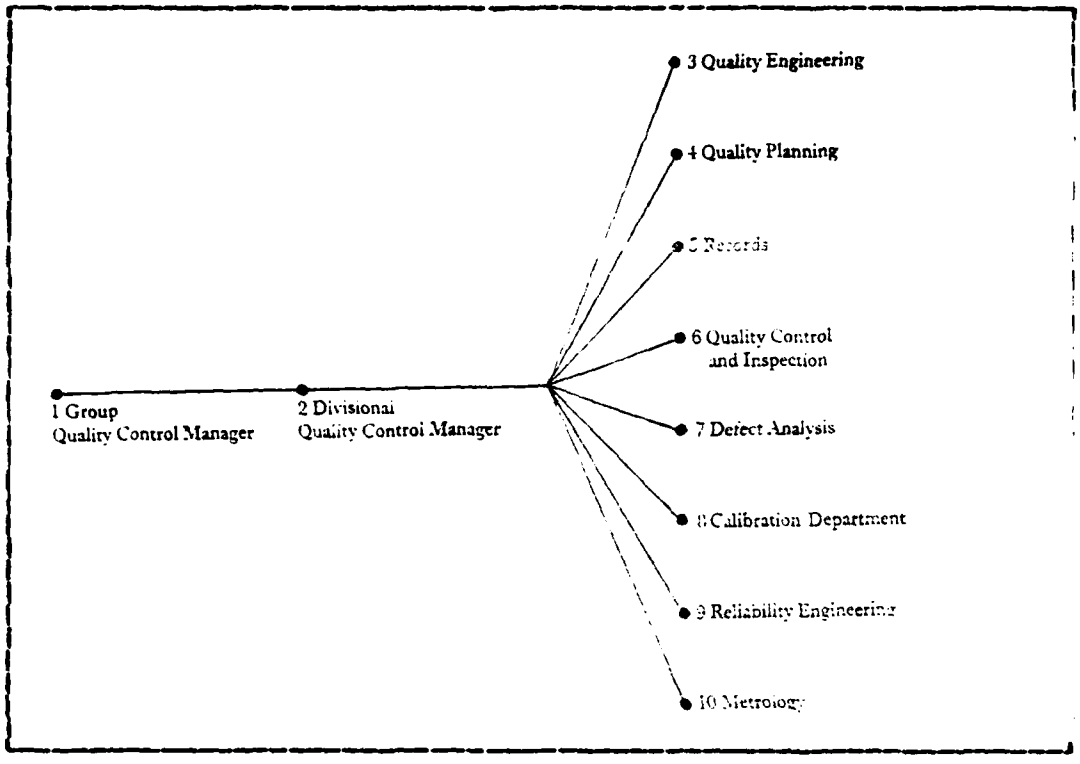


Figure 2.1 Typical Quality Assurance Organizational Structure.

software quality assurance techniques. Figure 2.1 illustrates a typical manufacturing quality assurance department.

B. THE ROLE OF QUALITY ASSURANCE

1. Why the Need for Quality Assurance?

The statistics concerning the growth of the software industry in the past 15 years as well as the problems concerning this growth are quite well documented and recognized. For example, one company, Boeing Aerospace, reported that on a large software project: a). only 14 percent of the total number of runs would have been required had there been no errors or failures, and b). 39 percent of the runs, while successfully completed, were later invalidated because of data errors, tape failures, or program bugs [Ref. 9]. From a cost standpoint, over eight billion dollars was spent for software of various types in 1980 [Ref. 10]. Computers, and in turn software, are becoming entangled in every aspect of our daily lives. From electronic banking and shopping to NASA space projects to more effecient use of farm machinery, we rely on computer software to help us make more and more of our decisions. Software developers have recognized their responsibility towards quality software and most data processing departments now incorporate some sort of quality assurance function into the production of software. This quality assurance function is able to alleviate the problem most software managers have of becoming involved in the system at a point when the cost becomes significant and the dates of implementation approach. The establishment of a quality assurance function provides management with a degree of confidence that an independent, technically trained group is monitoring the goals, methods and performance of applications from the beginning of the project.

2. Objectives of Quality Assurance

The quality assurance function, as part of the systems production group, works to ensure that standards concerning goals, methods and objectives are met. The quality assurance group typically performs those functions that the data processing manager might do personally if time permitted. Quality assurance reviews each system to ensure compliance wiht the following items. The system must meet the needs of the user department and other users and at the same time not infringe on the rights of other systems users. The goals of the system should be consistent with the objectives of the entire organization. If there is a conflict, the goals of the organization should maintain priority over the goals of one user. The system goals should also mesh with the EDP department objectives and if there is a conflict, it should be resolved before implementation. If there are external industry or government requirements, the goals of the system should conform to these standards. Controls on the system must be complete (management controls) and the system must be auditable. The system should conform to all general policies, procedures, standards and guidelines established by the organization and the electronic data processing department. Quality assurance must finally ensure that the design of the system is economical (least cost system), effective (desired results with minimum effort), and efficient (maximize use of people and machines).

3. Costs of Quality Assurance

The cost of a quality assurance function is very difficult to estimate or even measure. William Perry, an author of extensive material on software quality assurance and a member of the Quality Assurance Institute in Orlando,

Florida, comes closest to a precise figure by saying that "if quality assurance is included as a line item in a project's budget, it should range somewhere between 2.5 and 5 percent of the total project cost" [Ref. 4]. Figure 2.2 is indicative of the percentage of costs expended on a

DEVELOPMENT PHASE	% OF TOTAL PROJECT COST	
	Minimum	Ideal
Feasibility	.25%	.5%
Design	1.00	2.0
Programming	.75	1.5
Testing	.25	.5
Conversion	.25	.5
	2.5%	5.0%

Figure 2.2 Quality Assurance Project Costs.

typical five-phase software development project. This estimate is still rather idealistic because of the differences which may arise because of staffing alternatives, methodology, lifecycle entry, the difficulty in defining quality assurance and other unknowns.

Cost justification is an important aspect of implementing a quality assurance function. In the hardware arena, the cost of quality assurance is most often justified by lower warranty cost markups in the price of the product. This is equally true in the software world. Warranty costs will be lower for a quality software product. Another fringe benefit of a quality software product is its ease of adaptability to a similar product at a lower cost [Ref. 11].

The cost savings of an efficient quality assurance activity is often hard to justify because of the indiscipline of those not quality-conscious often makes measurement of improvements very difficult. As is usually the case, results speak the loudest. The cost savings involved in having projects done on time and within budget allows quality assurance to maintain its level of efficiency and reduces management time spent sorting out the mess which results from bad planning.

Finally, it has been shown that the entry point of the quality assurance function into the lifecycle of the project has a definite effect on the cost [Ref. 12]. The scope and structure of the quality assurance effort is affected strongly by the cost of errors related to the phase of development. Figure 2.3 is a typical illustration of the economics of error detection in the various phases of development. As this figure shows, the earlier a problem is detected, the less expensive is the cost of correction.

C. SOFTWARE DEVELOPMENT

1. Software Lifecycle Phases

Referring to the lifecycle of software is the most common method of addressing the development of a software product. A review of the liturature has produced a plethora of illustrations of what is considered to be the true or ideal lifecycle. Most examples use different terminology to describe what is happening at a particular stage in the lifecycle, but they all tend to include the critical items. The simplest lifecycle found is one in which there are only three stages - design and development, active and passive [Ref. 13]. Conversely, the most complex definition of a software lifecycle contains eight phases-systems definition, software allocation, specification, design, code,

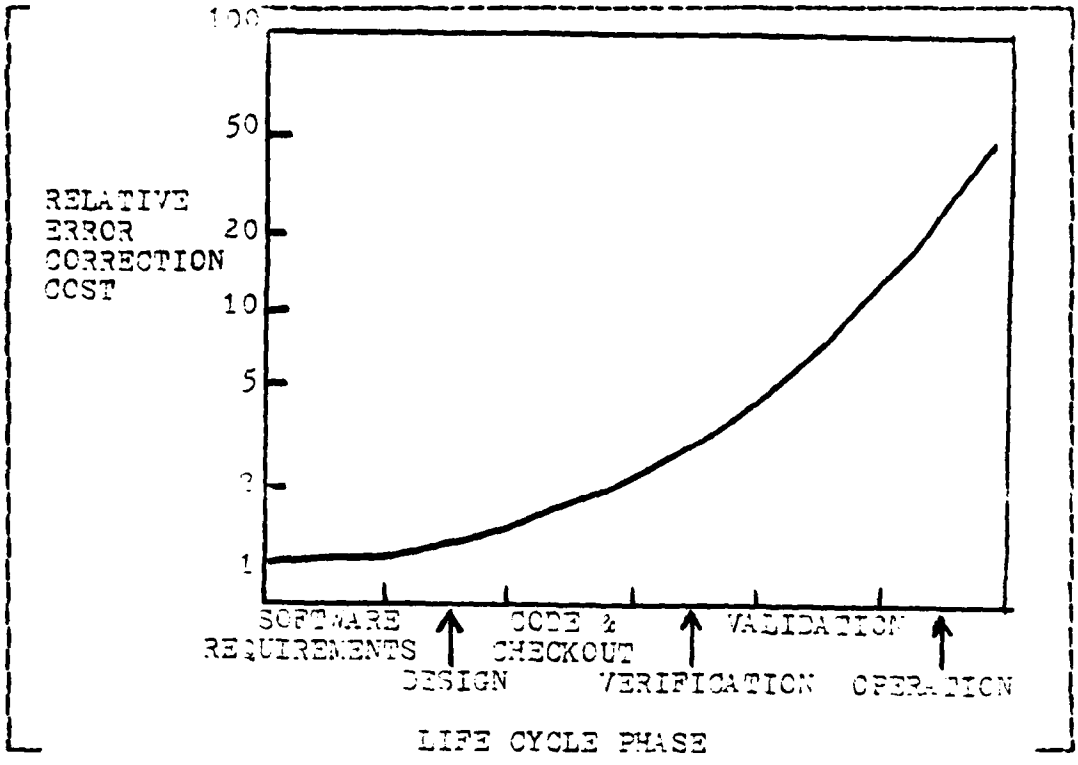


Figure 2.3 Relative Error Correction Costs.

verification, integration, and operation [Ref. 14]. Table 1 depicts a comparison of the various phases of the lifecycle of software.

Whatever the phases of software development are called, there are certain items which must be accomplished. The beginning of a project may be designated as a feasibility study, requirements definition, systems definition, user requirements study, initiation study or something else, but it consists of all activities which deal with determining whether or not a software project should be initiated. Such things as cost-benefit studies, goal definitions, and documentation requirements are typical activities which should be accomplished here. Next comes

TABLE I

Software Lifecycle Comparison

AUTHOR	LIFECYCLE PHASES
PERRY	Feasibility - Design - Programming - Testing - Conversion
FUJII	Conceptual Design - Requirements Definition - Design - Code and Checkout - Testing - Integration - Operational
MENDIS	Design - Code and Debug - Qualification Test
ROBERTS	Design and Development - Active Stage - Passive Stage
HOWLEY	Systems Definition - Software Allocation - Specifications - Design - Code - Verification - Integration - Operation
DUNN and ULLMAN	User Requirements - System Functional Specs - Software Functional Specs - Implementation - Verification and Test - Operations and Maintenance - Configuration Management
FIPS PUB 38	Initiation - Development - Operation

the general design of the system. This stage is normally labeled design, design and development, software design, or systems functional specifications. Activities such as design alternatives, specific requirements, functions to be performed, and program and data base specifications should be included in this phase. The next phase is probably the most rudimentary in terms of work and deals with programming and testing. This stage is also referred to as coding and debugging, verification and validation (after coding is complete), or is sometimes broken into two distinct phases. The system is now written in the desired language and various tests are performed to ensure the system performs as desired. The next, and most often last phase, is referred to as conversion, integration, operations, implementation, maintenance, or configuration management. This stage

consists of maintaining the software, performing ongoing evaluations and changing it as additional requirements are identified.

2. Software Properties

As stated previously, software is an elusive product upon which to place a quality measurement. Howley and Fink, software quality engineers for the Boeing Aerospace Corporation, have attempted to verbalize what a quality software product is by stating "A quality software product may be defined as one which exhibits the following properties: it satisfies the software specification and design requirements; it performs all intended functions; it is relatively free of design, interface and coding deficiencies; it has a low life-cycle cost; it is properly identified and documented; and it incorporates all needed software quality characteristics" [Ref. 15]. To achieve the level of quality which is desired by the above definition, there are a number of factors which contribute to the production of quality software [Ref. 16]. These include, but are not limited to:

1. Correctness - This generally means programs perform in exactly the manner specified in the program documentation. Correctness is usually considered an ideal quality which is rarely achievable.
2. Reliability - This attribute means that programs perform relatively trouble free all the functions expected from the specifications or documentation.
3. Validity - Validity is concerned with the question of whether the functions and performance of the programs are adequate and suitable to a needed purpose. The software, without manual intervention or additional programming, should perform the functions that reasonably would be expected of it. This attribute

is a very subjective one and must be flexible to changing requirements.

4. Resilience - This means that programs should be designed in such a way to be forgiving of common user and data errors. Inconsistent or unacceptable data entries shouldn't provoke actions which make no sense to the user.
5. Usability - Human factors and limitations and convenient usage techniques should be considered whenever a program is written.
6. Clarity - Programs should be easily understandable from the users manual and all the documentation should be clear, concise and cogent. Programs should be modularly designed, have explanatory comments where necessary, and use meaningful choices of variable names.
7. Maintainability - Good documentation and comments as well as clear structure will make programs more easily repairable. Clarity is also essential for making minor improvements.
8. Modifiability - Major changes should be anticipated and the software designed so that program functions that might require major change are well documented and isolated in distinct modules.
9. Generality - Programs should be applicable to a wide range of input values and usage modes.
10. Portability - Programs should be easily adaptable to transfer to another computer system or operating system.
11. Testability - Programs should be simply structured and use generaly algorithms, to facilitate step-by-step testing of all capabilities.
12. Efficiency - The attempt should be made to keep the cost of program operation as low as possible.

The preceding list contains many of the attributes of a quality software product but is not the only list available. Other authors include such things as integrity, flexibility, reusability, interoperability, and others which are descriptors of quality software. Figure 2.4 is a good illustration of how these factors affect each other and what degree of a certain factor is required when a different factor is recognized. As can be seen, some factors are synergistic while others conflict. The impact of conflicting factors is that the cost to implement will increase. This will serve to lower benefit to cost ratios [Ref. 17].

3. Hardware Characteristics vs. Software Characteristics

Hardware is a tangible piece of engineering. It has very precise specifications and drawings and is based on well established building principles, the aim being to manufacture many identical (or near identical) items. The design-development-production cycle is mature and well tuned. Refinements to the design may be made many times before a commitment to manufacture is made. In contrast, software engineers ship their prototypes. Software is a largely intangible product, only described by many volumes of specifications and listings. Software is unlikely to go through as many prototype stages and, therefore, the opportunity for design iteration and improvement is somewhat limited [Ref. 18].

Most aspects of hardware are functionally testable and have very specific requirements testing programs. It is fenced-in by established principles and well-known, widely-used disciplines. Unlike hardware, software is functionally non- testable in all but the simplest of computer programs and as a result, it is very difficult to test software

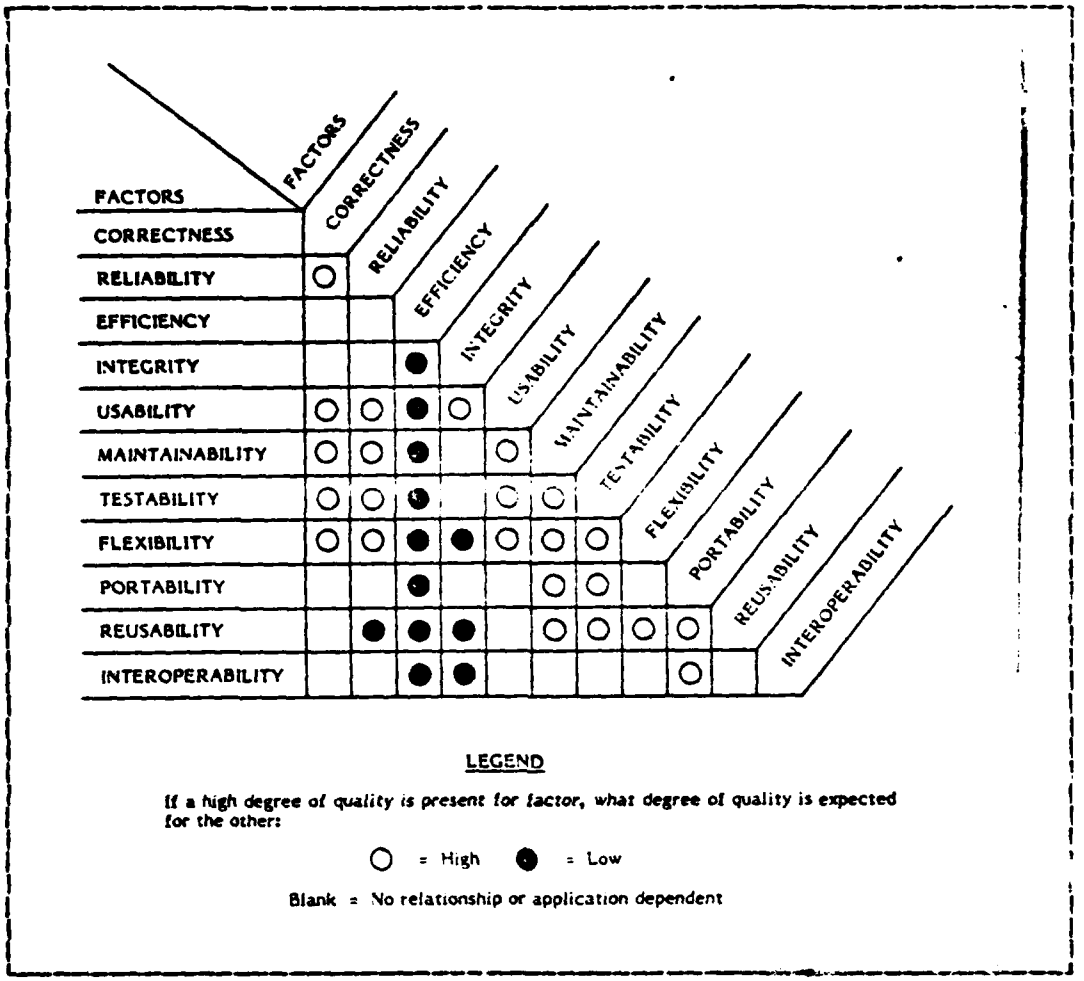


Figure 2.4 Relationships Between Software Quality Factors.

completely [Ref. 19]. Figure 2.5 illustrates the problem of testing software. Each circle represents a processing node. The clockwise arcs are jumps around the individual nodes, and the counter-clockwise arcs specify the number of iterations of each half. The number of discrete states possible within this trivial diagram is approximately 100 quadrillion.

If these could be tested at the staggering rate of one per microsecond, it would have been necessary to start the

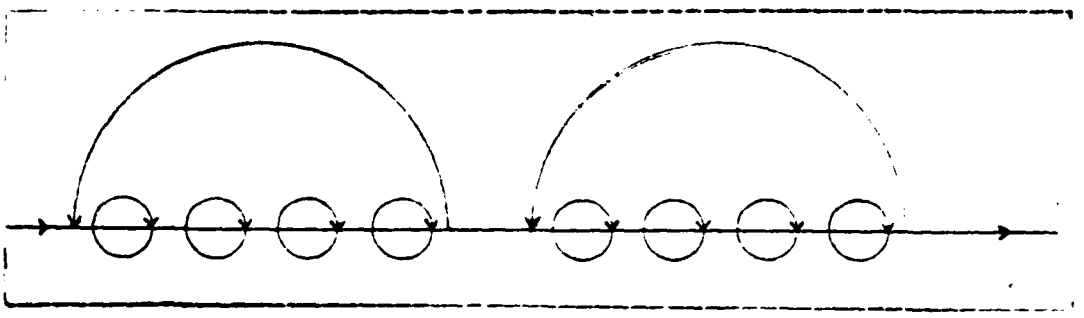


Figure 2.5 Computational Paths.

testing over 2,000 years ago to meet next month's scheduled delivery.

The usual cause of hardware failure is component deterioration. Software failures are almost always design errors that show up only when the software is used under certain conditions. Hence, Quality Assurance techniques for software focus on getting the design right [Ref. 20].

A comparison of hardware and software lifecycles is offered by Table 2 and shows clearly that similar terms in the two fields sometimes have radically different meanings.

4. Software Management

The driving forces behind implementing management structure in any organization are reduction of costs, increased control, and production of a quality product. Software production is no different. Figure 2.6 depicts the rise in software costs in the last twenty years. As is apparent, software costs greatly exceed equipment costs over the useful life of computer services. With this kind of growth it is imperative that software be managed to minimize costs. While cost minimization is an important aspect of software management, there are other reasons of equal importance. Most software in production today is a complex

TABLE II

Hardware and Software Product Life Cycles

	HARDWARE	SOFTWARE
DEVELOPMENT	Determine user Requirements Develop Product Concept (Functional) Specify Component Design (Detailed) Build and Test Prototype Develop Manufacturing Techniques	Determine user Requirements Develop Product Concept (Functional) Specify Component Design (Detailed) Implement and Test Programs --
INSTALLATION	Manufacture Product Make Product Available to Users	Copy Programs Make Program Available to Users
MAINTENANCE-IMPROVEMENT	Maintenance (Correct Component Failures) Recall Product to Correct Design Flaws Enhance Product	-- Maintenance (Correct Implementation and Design Errors) Maintenance (Adapted to Changed User Environment)
PHASE-OUT	Unit is Unusable and Unrepairable (Replace) Product is Obsolete	-- Product is Obsolete

technical activity that must be directed effectively. The complexity of any program in all but the simplest of applications is such that programming has evolved past a routine effort that can go unsupervised or be done by junior personnel. Any investment of funds or resources is likely to be a major one for any organization and the technical choices may have widespread effects throughout the organization. Management and technical control by professionals is essential for resolving design issues and giving adequate direction to programmers regarding cost and schedule targets. Software management also provides for

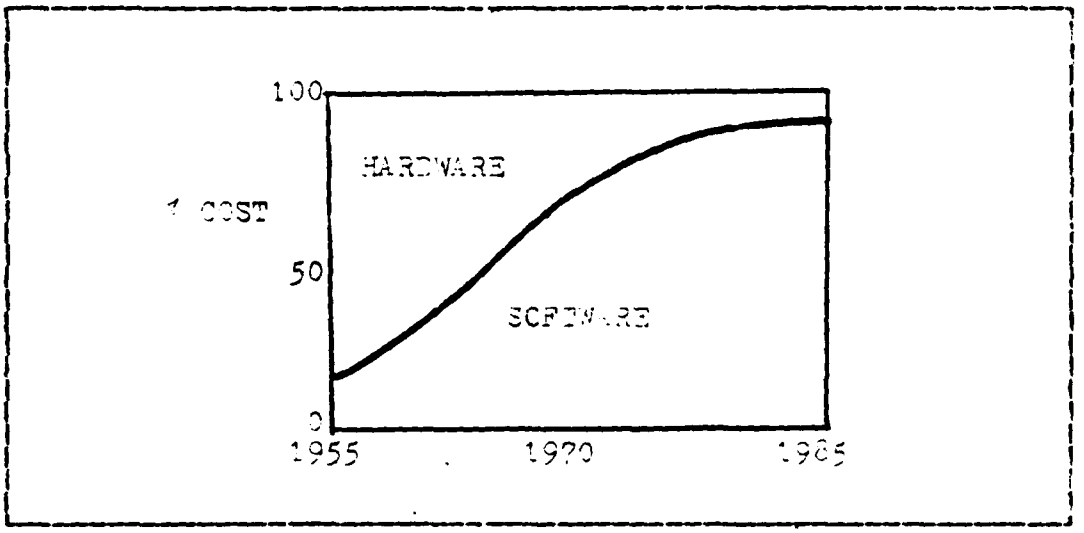


Figure 2.6 Estimated Growth of Software Costs.

accountability of project decisions and objectives and gives top management visible measures of success in the accomplishment of goals. As a general rule, software projects are often initiated by management personnel who control budgets and schedules and software designers frequently end up doing a substantial amount of independent work with little pressure to evaluate their progress or the remaining work or costs. With a software managment structure in place, important issues and objectives such as cost, quality and schedule can be carefully evaluated and the appropriate responsibility for the decisions assigned. Technical controls, working procedures and resource management are further justifications of a strong software management structure. Questions about system feasibility, system quality, design methodology and testing procedures are ones which should be answered by software management. All of these things help designers and programmers to organize and direct their efforts efficiently in solving such problems

within available cost and time limits. Another reason for software management is the ongoing problem of maintenance after implementation. Some estimates put the cost of maintenance at 70% of total software costs. It is important to recognize that maintenance is just as important as development. Good software management principles will attempt to be ongoing throughout the lifecycle and will make the strongest effort to stress close management of effort toward the most needed software capabilities.

The management of software development is often referred to as "software engineering". This implies that the principles of production engineering management can be transferred or applied to software development. Software engineering suggests that "the entire development of a software product from initial conception through design, implementation, testing, and maintenance can be organized in a systematic and manageable fashion. It should, therefore, be possible to monitor the quality, performance and cost of the end product through the several phases of its life cycle" [Ref. 21].

5. Standards for Software Development

An area which has been seriously neglected in the software development industry during its growth has been the establishment of standards of conformance. There has been a recognition of this lack during the past few years and attempts have been made to provide adequate standards. The most widely used military document concerning standardization of quality assurance plans is MIL-S-52779A dated 1 August 1979. This document is applicable to Department of Defense agencies when acquiring software where the acquisition involves either software alone or software as a portion of a system or subsystem. It provides specific guidance concerning software quality assurance program requirements

and covers such things as tools, techniques and methodologies, computer program design, work certification, documentation, computer program library controls, reviews and audits, configuration management and testing [Ref. 22].

This document is used not only by DOD, but has also been referred to by many civilian organizations in the absence of anything better. The Institute of Electrical and Electronic Engineers has recently sponsored a committee to evaluate the problem and develop a set of software quality assurance standards. Their stated purpose was to "provide uniform, minimum acceptable requirements for the preparation and content of software quality assurance plans" [Ref. 23]. The sections of the standard developed contains direction concerning such things as reference documents, management, documentation, standards, practices, and conventions, reviews and audits, configuration management, problem reporting and corrective action, tools, techniques, and methodologies, code control, media control, and supplier control. As can be seen, it is extensive and comprehensive in scope and provides guidance for development of a thorough software quality assurance plan.

The preceding two documents are the most widely used standards referred to when developing a software quality assurance plan. There are some other publications which can be reviewed for direction concerning development of software. Federal Information Processing Standards Publication 38 (FIPS PUB 38) provides thorough and comprehensive guidelines for documentation of computer program and automated data systems. National Bureau of Standards Special Publication 500-11 is a good overall guideline to computer software management and quality control. There are a host of journal and proceedings articles dealing with software quality assurance which provide information. All of these are not "official" guidelines and lack any authoritative

endorsement. The best hope for better results from software quality assurance programs lies in an acceptance of the IEEE Standards and conformance to their requirements. Unfortunately, these are relatively new and there is insufficient data to declare that new industry standards have been developed.

D. SOFTWARE QUALITY ASSURANCE METHODOLOGY

1. Quality Assurance Planning

Planning is essential for the successful achievement of any project and must remain dynamic to be of any use. It is important that plans are modified to reflect changes in requirements as they occur. On a broad scope, a quality assurance plan must indicate the particular activities which will enable the required level of quality to be achieved on any given project. How the product is to be assured and what activities the quality assurance group is to undertake in order to satisfy organizational requirements are key elements.

Quality assurance generally parallels the systems development process. The position of review points will depend upon management's requirements concerning decision points or information requirements. The importance of the system to the organization as a whole will determine the amount of time spent on each project. The critical points are the end of each systems development life cycle phase. At this time, an opinion is rendered by quality assurance as to the adequacy of the design process up to that point. This opinion can then be used as a decision making factor in determining whether to progress to the next phase. In discussing the review criteria a five phase systems lifecycle will be assumed.

The feasibility study commonly consists of evaluation of alternatives and techniques to solve a particular problem and recommend a course of action to management. This may or may not include a computer system and the personnel involved in the study may or may not have computer experience. The role of quality assurance during the feasibility study is usually one of a consultant to discuss the practicality of alternatives or cost estimates. At the end of the feasibility study, quality assurance should evaluate whether the study team followed the organization's procedures in developing a proposal for management and comment on any computerization aspects of the proposal.

The next phase, design, is critical to quality assurance. The greatest impact is made during this phase and the quality assurance group should strive to impact the design without actually participating in the design process. The goal is to not argue for or against particular designs but to review the proposed design on merit. One method which is widely used in this phase is to divide the design into two phases-informal and formal. The informal phase occurs after a preliminary design is on paper and consists of quality assurance giving discussion only review to allow the design group to determine if they are on the right track. There is no report to management generated from this phase. A structure such as this requires a good working relationship between quality assurance and the design group. At the end of the design phase, a formal review occurs and the design is normally fixed at this point. Compliance to performance criteria, systems goals and procedures are reviewed by quality assurance.

Program design and program coding make up the next phase which must be evaluated by quality assurance. While these two activities may be combined into one phase, it is usually more effective and it facillitates structured

programming to separate them. By reviewing the program design, quality assurance has a greater opportunity to ensure compliance to design procedures and standards. This will hopefully alleviate many problems usually encountered in the coding phase. At the end of the programming phase, quality assurance should perform a review to ensure compliance to procedures and standards for such things as coding and use of operating system facilities. This should be a detailed review and quality assurance must examine all aspects of coding, operating system instructions, file structures and anything else which will affect the operation of the computer.

In the next phase, system testing, quality assurance is mostly concerned that an adequate test plan has been prepared, that it is followed and that it conforms to the standards of the organization. Quality assurance will only review test results to ensure compliance to standards and should not become involved in a detailed test plan. At the end of the system testing quality assurance should again review for conformance to organizational policy and check for user satisfaction.

The last phase in our example, implementation, can be the broadest in scope and longest in duration. This phase is sometimes called conversion and is generally thought of as the process of replacement or new installment. As with testing, the primary concern of quality assurance is that a bonafide plan has been defined and that it is being followed. The completion of the implementation phase brings the final review by quality assurance that the procedures defined in the design stage were followed. Once again, user satisfaction is of paramount importance and quality assurance is reviewing plans and procedures.

The foregoing example of a quality assurance plan over the lifecycle of a software product is by no means the only one to be used. Another typical example is discussed by Marilyn Fujii, a software quality assurance professional from Logicon, Inc., in which the lifecycle is divided into seven parts and quality assurance again has a role over the entire lifecycle [Ref. 24]. The early stages of the plan are centered around defining the procedures and standards which will be applicable to supporting configuration management and computer program development. Most other activities consist of reviewing and auditing software products against previously set standards. Quality assurance is responsible for all design reviews and audits and they evaluate all documentation such as test plan, specifications, and users manuals. Any walkthroughs or acceptances testing will be scheduled and conducted by quality assurance. At the delivery point in the lifecycle, they audit the final configuration to be installed in the operational environment. Figure 2.7 offers a visual presentation of quality assurance's role in this example.

The examples given are but two of a multitude which can be found in professional literature. Regardless of what specific method is used, there are a number of components which should be included in any software quality assurance plan. The plan should identify procedures to be used in issuing work tasking instructions for all work relating to software development. Monitoring of procedures and assuring adherence to them should be part of any plan. Identification of schedules and resources and tracking progress toward them should be included. Work descriptions, responsibility assignments, initiation procedures, report generation procedures, and scheduled completion dates should be addressed. The plan should document quality assurance involvement in the specified development program. Also

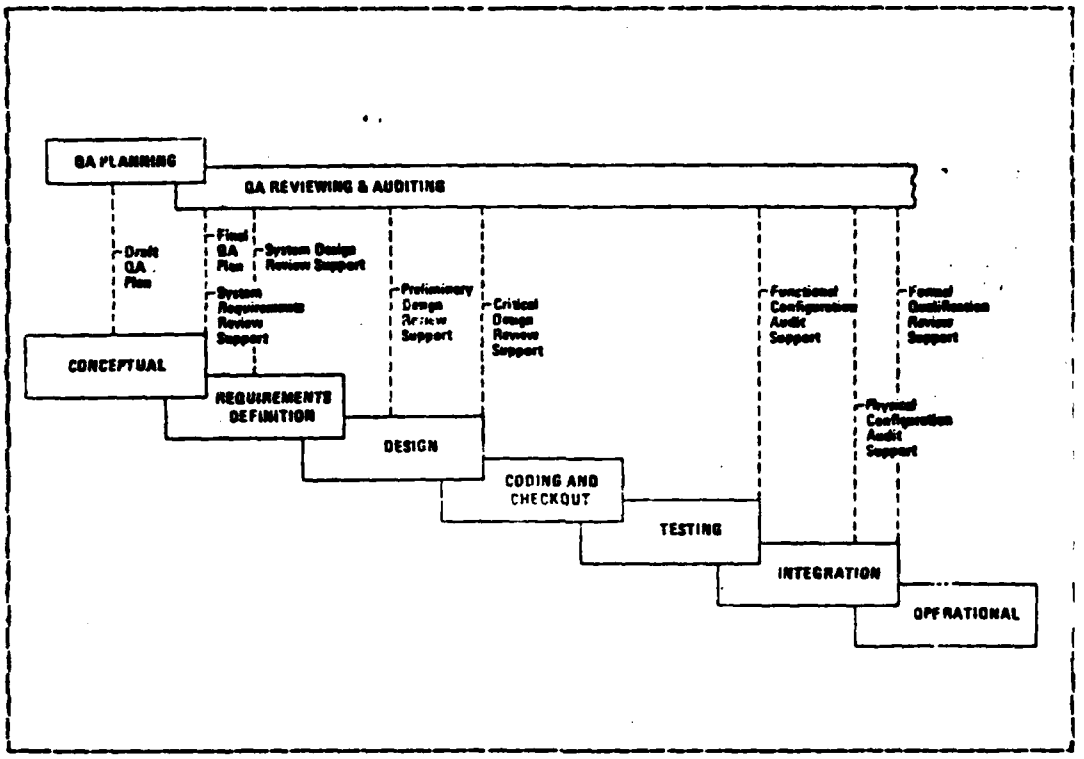


Figure 2.7 Quality Assurance Activities.

provided should be visible schedules, milestones and interdepartmental dependencies and commitments. Levels of detail required should be included in any plan. Finally, the organization responsible for software quality assurance should prepare the plan. A general outline of a typical quality assurance plan is provided as Figure 2.8,

2. Staffing and Organization

The success of any quality assurance function begins with the personnel assigned to the staff. Individuals as knowledgable as senior systems analysts and designers should ideally be assigned to quality assurance. It is not enough, however, for quality assurance personnel to merely possess

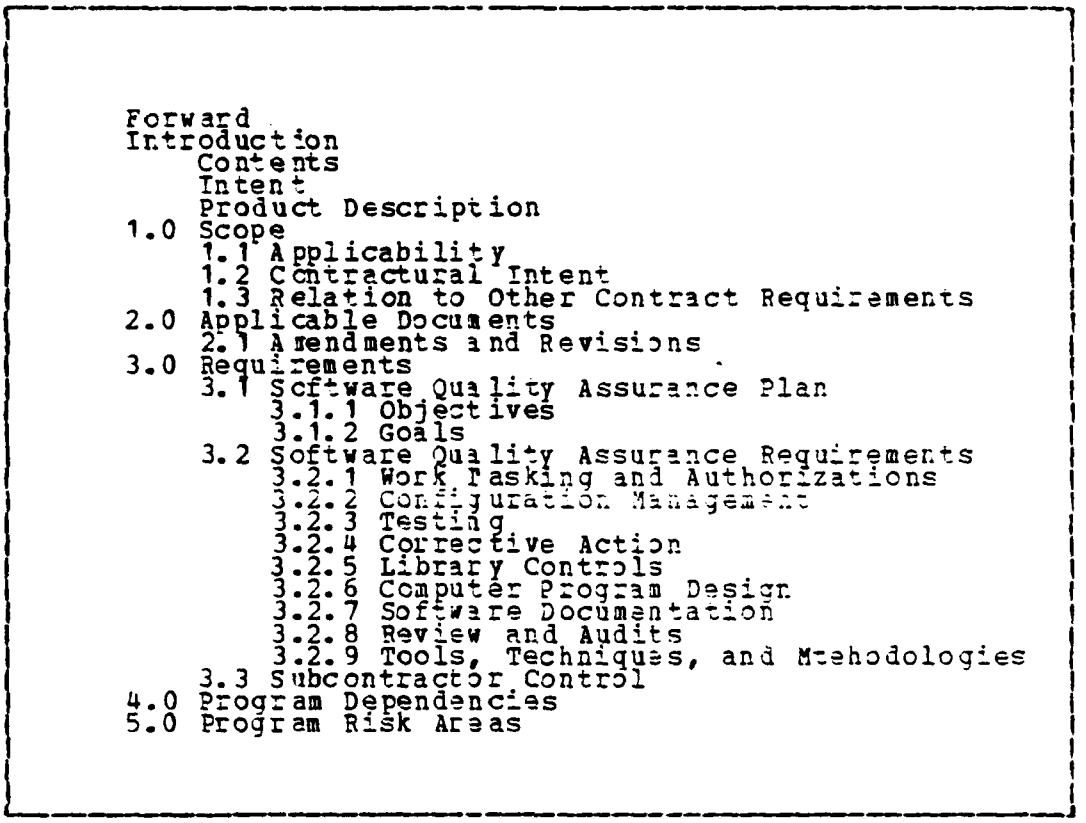


Figure 2.8 Typical Software Quality Plan Outline.

the characteristics of a good systems analyst. Quality assurance personnel must command the respect of both the individuals whose systems are being evaluated and the management of the EDP department, who must rely on them. The ability to review the work of others and to convince them there are better methods to perform their work takes some unique skills in dealing with people. Quality assurance reviewers must have a talent for good communicative and persuasive ability as as well as be respected for their technical ability.

There are a number of variables concerning the experience and number of people assigned to quality assurance. The size of the project at hand is a major determining factor. Small projects which appear to be relatively simple and perhaps repetitive of previous jobs may be short-changed in the quality assurance area. If the company itself is small, it may not be able to afford the commitment to quality assurance of a large corporation. Top management may not recognize the need for quality assurance and hence give it less than prominent attention.

The organization of a quality assurance department can be set up in many ways. One widely held theory is that the higher in the data processing structure the quality assurance function reports, the better the probability of success. Also, the level of reporting is sometimes indicative of management support [Ref. 4]. Figure 2.9 shows quite a simplified view of a representative EDP department with the quality assurance function placed as a staff function reporting to the EDP Manager. This structure insures that quality assurance will receive the attention of the EDP Manager and that it will be independent of all other aspects of the data processing department. Figure 2.10 shows an organizational structure from industry- Informatics Inc. Quality Assurance in this organization is embedded in the organizational structure and has secondary affects throughout the company. As another example, Figure 2.11 shows a variation of the placement of the quality assurance function [Ref. 25]. This structure, with quality assurance as an independent function reporting directly to a division general manager, provides good independent oversight.

Different projects within an organization will receive different emphasis in the quality assurance area. The same holds true for industry as a whole. Embedded software in a weapon system will not receive the same sort of

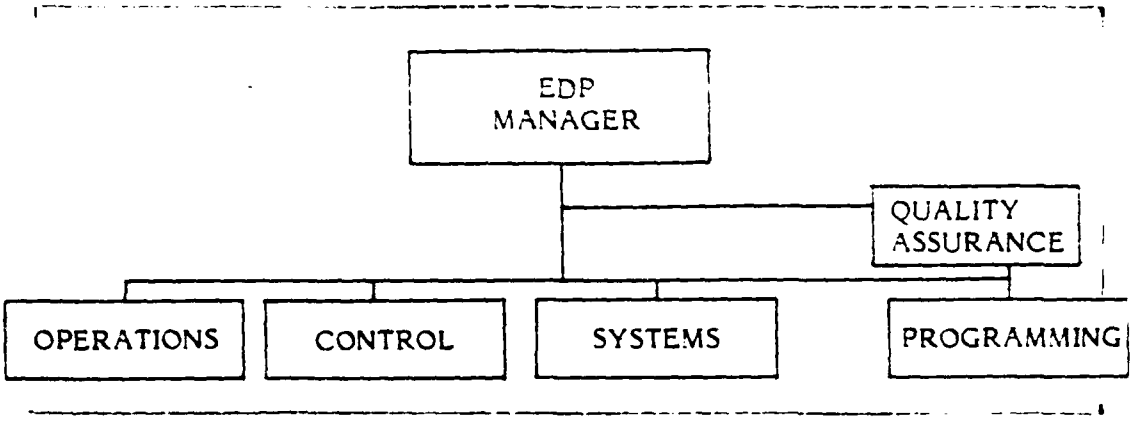


Figure 2.9 Sample EDP Department Organization.

quality assurance attention as a payroll project. As each project has differing requirements, so will each quality assurance scenario be different. Throughout literature concerning quality assurance implementation, there are a number of methods used to organize a quality assurance function. These have been consolidated into four general methods and will be described in some detail. It must be remembered, however, that the type of organization of the quality assurance function will depend greatly on such factors as the type of product and industry, emphasis given quality assurance within the organization and the scope of the project. Quality assurance departments from selected companies will be examined in detail further in the paper which will expand on the methods of organizing the quality assurance function.

The first method, and probably most widely used when organizations are beginning the quality assurance function, is the task force method. This method allows organizations to become involved in some sort of quality assurance activity prior to the formalization of the quality assurance function. A task force offers the advantage of developing a

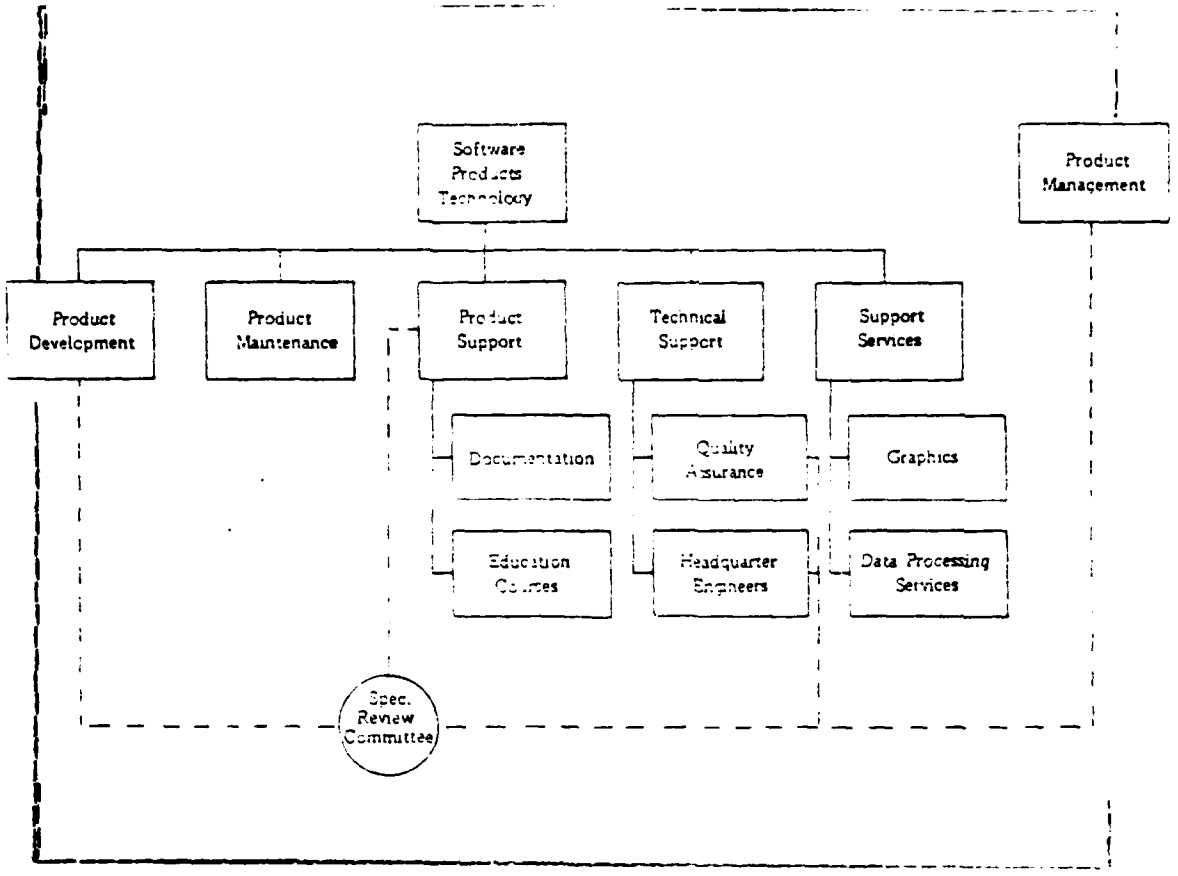


Figure 2.10 Informatics Inc. Organization Chart.

group able to handle the unique problems which may be encountered in a given software project. Task force members with the appropriate background can be hand-picked by the EDP manager. Another benefit is the training afforded the systems designers assigned to the team because it puts them in a position of analyzing the competency of systems design. A disadvantage to this method is the problem of continuity. Each task force will tend to develop its own methods and procedures for the review. If the task force members are not relieved of a significant amount of the burden of their daily work, then another possible problem is that they may have trouble finding adequate time to devote to the review.

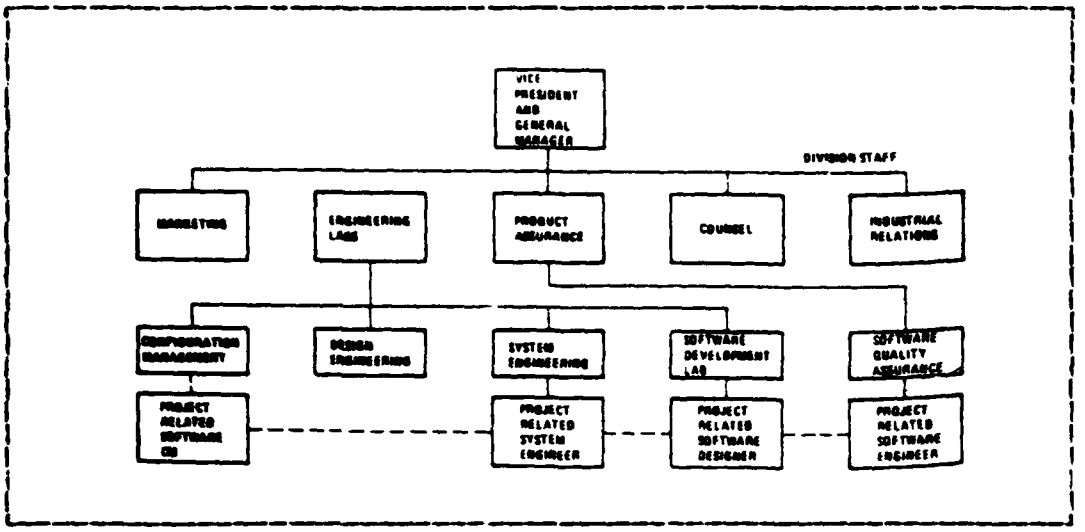


Figure 2.11 Traditional Organizational Structure.

A second method is the formation of a full time quality assurance staff. This method provides the greatest amount of continuity among reviewers. The EDP manager can thus have a greater degree of confidence in the quality assurance function. By assigning a full time staff, management is giving a signal of the measure of importance it places on quality assurance. The biggest disadvantage of a full time staff is the competency of the review group. Whereas a task force can add specialized knowledge, a full time staff operates with the personnel assigned. Another problem is the technical proficiency of the staff. Technical proficiency with current practice is very important both from the standpoint of credibility of the staff with the rest of the organization and the proficiency with which the personnel can perform their function.

The permanent committee method is another approach and is basically just a step up from the task force method. Continuity of individuals is the biggest difference between

the two methods. Where a task force reviews one project, a committee will be convened for the purpose of reviewing many projects. This says to project managers that their projects will be reviewed. The permanent aspect of the committee indicates a higher degree of management support than a spacially convened task force. As with a task force, a permanent committee has the problem of the amount of time reviewers can devote to projects under review while still maintaining their workload. Another negative aspect is that it still is just a committee and will lack the authoritativeness of a function staffed with full time personnel.

The fourth method is a combination of full time and part time personnel. This method can be accomplished with the use of one or more full time personnel to maintain continuity of the quality assurance function and augmented by part time personnel to assist in reviews. The obvious advantage would be the ability to add specialized knowledge as needed to review projects. If this method is used, one individual should be named to head the quality assurance function. He should be a strong personality with supior knowledge of the requirements of of a quality assurance function and possess the ability to direct part time personnel in the most effecient use of their time. It is important that the manager be on equal footing with other line functions and can insist that only the best people be assigned to quality assurance.

3. Reviews and Audits

Reviews are conducted sequentially throughout the lifecycle in order to facillitate transition into a subsequent developmental phase. As previously stated, reviews should occur at the completion of each development phase. Perry [Ref. 4] has further identified twelve review points which will not only review but influence systems design as well. These reviews occur at the following points:

1. Midjustification phase,
2. End of justification phase,
3. Business system solution phase,
4. Computer equipment selection,
5. Computer system design,
6. Program design,
7. Testing and conversion planning,
8. Program coding and testing,
9. Detailed test plan,
10. Test results,
11. Detail conversion planning and programs, and
12. Conversion results.

Naturally, the number of review points would depend on a number of variables including size of system, impact on the organization, and makeup of the quality assurance organization. In addition to these review points, quality assurance can perform valuable consultation while conducting the reviews.

Most authors are not as specific as Perry as to the timing and placement of review points. The general consensus is that reviews must be predefined, occur at key points in the development process, be understandable and thorough, and are conducted in accordance with prescribed procedures.

There are a number of techniques that a quality assurance review team may employ during the course of the review. When gathering information about the system being reviewed, such things as project documentation, system documentation, interviews, observations, and the use of established checklists are appropriate methods of gathering information. Practices used when attempting to validate the information given during the gathering phase inculde testing, evaluating test data, formulating base case data, and individual confirmation. After the information is

gathered and validated quality assurance must evaluate the data for management. This evaluation is typically based on intuitive and evaluative judgement, mathematical simulation or modeling, expert advice, or quantitative analysis. This list is not exhaustive, but is representative of the types of techniques used by quality assurance reviewers in achieving fair and comprehensive reviews.

Auditing is sometimes differentiated from reviews. Audits are usually thought to be final acts where all loose ends in a quality program are tied up. Types of audits include in-house audits where the audit verifies that the developer is adhering to all development standards and procedures, subcontractor audits to ensure that the subcontractor is complying with all software standards and procedures imposed by the contract, and fact-finding audits in which the subcontractor is evaluated to ensure he is capable of furnishing reliable, quality software of the type deemed necessary to meet contractural requirements.

The Institute of Electronic and Electrical Engineers standards propose a certain minimum number of reviews which should be conducted during the software development life-cycle [Ref. 7]. These include a software requirements review to ensure the adequacy of the requirements stated in the software requirements specifications, a preliminary design review to evaluate the technical adequacy of the preliminary design of the software, and a critical design review to determine the acceptability of the detailed software designs. Recommended audits consist of a functional audit which is held prior to software delivery to verify compliance to all requirements specifications, a physical audit to verify that the software and documentation are internally consistent and ready for delivery, and in-process audits to verify consistency of the design.

4. Testing

Verification is another word for testing. It is essentially ensuring that the conditions are as stated. It involves doing whatever is necessary to verify that the statements or conditions are correct. Although correctness is the overall goal for most testing efforts, it is not always the overriding concern. Large programs are sometimes so complex that they never completely satisfy their specifications. These programs may be quite usable because failures are encountered infrequently in practice, and when they do occur, their impact on a user is small. To be usable, correctness is not always necessary and sometimes not good enough. A correct program may satisfy a narrowly drawn specification and yet not be suitable for operational use because, in practice, inputs not satisfying the specification are presented to the program and results of such incorrect usage are unacceptable. Thus, if a program is correct with regard to an inadequate specification, its correctness is of little value. The problem which arises is that most testing consists of correctness tests. There is very little testing done for reliability, robustness, efficiency, and other properties which make a valuable software product. Whatever property is being tested, the tests which are valuable are those where the result is not predictable, so that application of the test and acquisition of the result constitutes an information gain or a reduction in uncertainty [Ref. 26]. To achieve this goal, tests should check the program at the boundaries of its behavior. In order for software to be tested in the most effecient manner, a test plan with complete procedures and methodologies must be formulated. Other than from the obvious reason of ensuring test efficiency, the reasons for this are to provide an audit trail of testing so that future problems

may be dissected from the point they initially surfaced and that boundary areas where testing would be most efficient may be easier to identify.

Software quality assurance should become involved in testing in a number of areas as illustrated in Figure 2.12 Before the testing begins, it should ensure that all software, hardware, and the environment are in a satisfactory state and that test and simulation software have been defined and are under control. It should witness loading and running of the software and ensure the test results are retained and all discrepancies noted. Finally, quality assurance should assist line functions to ensure determinations concerning deviations and discrepancies are recorded.

MIL-S-52779A, the standards for software development used by the Department of Defense, contains a comprehensive list of software testing procedures [Ref. 22]. These testing procedures are utilized by many civilian software development organizations and consist of the following: (a) analysis of software requirements to determine testability, (b) review of test requirements and criteria for adequacy, feasibility, and traceability and satisfaction of requirements, (c) review of test plans, procedures, and specifications for compliance with contractor and contractual requirements and to insure that all authorized and only authorized changes are implemented, (d) verification that tests are conducted in accordance with approved test plans and procedures, (e) certification that test results are the actual findings of the tests, (f) review and certification of test reports, (g) ensuring that test related media and documentation are maintained to allow repeatability of tests.

Before Test

1. All software and hardware under control
2. All test and simulation software defined
3. All facilities available

During Test

4. Witness loading and running
5. Record discrepancies
6. Identify and retain output and results

After Test

7. Participate in Analysis
8. Raise outstanding points as discrepancies
9. Retain analysis results
10. Certify test report on satisfactory completion

Figure 2.12 Typical Functions of a QA Dept. During Testing.

5. Software Quality Assurance Tools and Techniques

Improving software development and test processes depends in large part upon the application of proper tools and techniques to the development lifecycle. The differentiation between tools and techniques is very clear. A technique may be defined as a procedure for implementing a reliability or quality goal. Techniques consist of standards and procedures used in development and maintenance of software systems [Ref. 25]. Such things as structured programming, top-down design, system modularity, proper language selection, abstraction, information hiding, and program design languages are generally thought to be techniques in achieving software quality.

Tools, on the other hand, have been defined as an automated technique [Ref. 25]. Computer programs which perform measurement tasks which would otherwise have to be done manually are considered tools. There are a large

number of tools to ensure software quality assurance on the marketplace. The problem is that most automated testing tools are expensive to install and use, different tests require very different tools and most tools are incompatible with each other [Ref. 27]. Selection of specific tools should be done only after careful analysis of the objectives desired of the tools, the tools cost- funding and the criticality of the software functions to be tested. Another consideration should be the phase of development which the software is in. Figure 2.13 shows some typical tools which

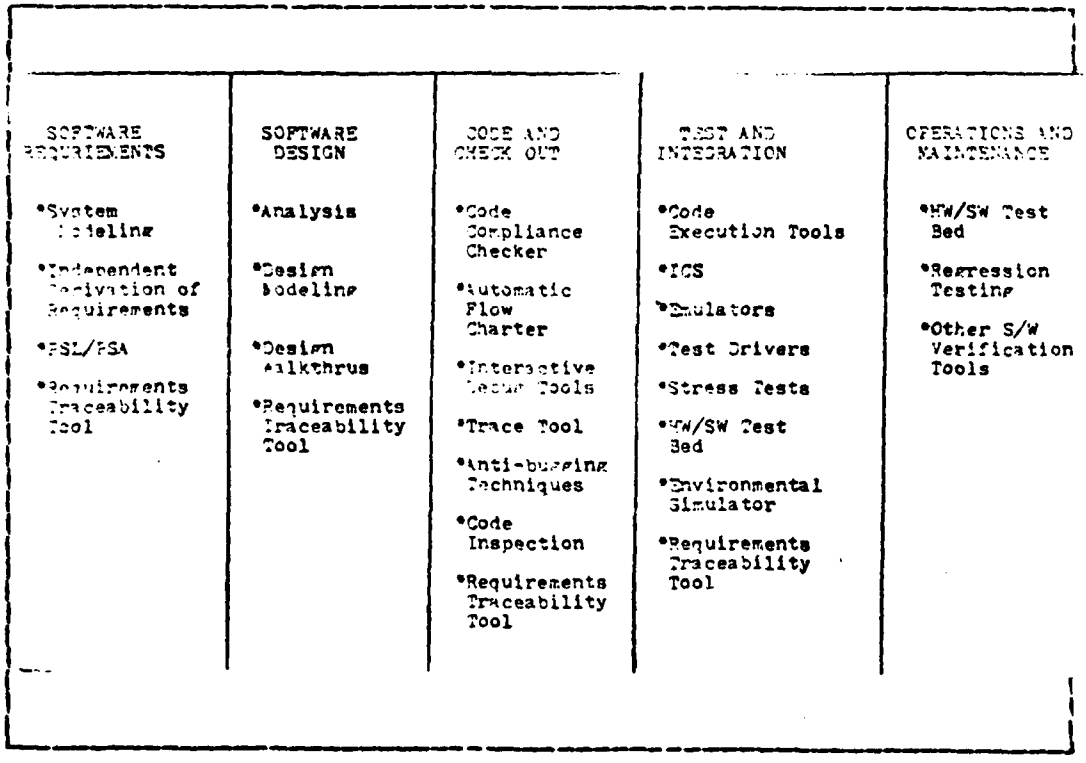


Figure 2.13 Tools Used by Software Phase.

may be used during the lifecycle of software development. While not a specific list of tools which may be utilized, the figure gives an idea of the types of tools found in

industry today. A brief explanation of selected tools from the above list follows. System modeling--a technique whereby a simulation of the hardware/software system is programmed using a simulator. Interactions between hardware, software, and personnel are simulated and incompatible system requirements often become evident after system modeling. PSL/PSA (Problem Statement Language/Problem Statement Analyzer)--This is a specific tool licensed by the University of Michigan, Project ISDOS. It provides a means for describing information, computer and software systems. A requirements data base built from several contributers can be checked for consistency and formal completeness. Design modeling--Critical algorithms are coded in a representative manner to determine of the design will result in the desired accuracy and execution times. Timing, memory margins, resource utilization, and traffic rates are modeled to ensure adequacy. Requirements traceability tool--Software requirements are linked to successive design data base entries, test planning, and test data entries to provide requirements traceability. Interactive debug tools--The debug tool controls the code while it is executed and displays memory and registers. The registers and memory can be displayed while the code is executed instruction by instruction. Preset memory locations and registers hold any desired value thus allowing branches to be executed and the logic debugged. ICS (interpretive computer simulation)--This tool allows the instructions, interrupts and input/output capabilities to be made visible by simulating the architecture and memory of a larger computer. The program can be started and stopped in order to evaluate performance of the program at various points. Stress tests--As the name implies, this tool tests the computer under worst case conditions of various parameters such as memory input rates, memory utilization, etc.

Hardware/software test beds--A test bed is the system level hardware joined with the system software and combined with the appropriate test drivers, monitors and environmental simulators to provide as near an operational system as is possible. Regression testing--Regression testing uses a standard proven test for testing the software after a change has been incorporated in the software in order to detect any side effects or errors due to the change. These tools are but a few of the literally hundreds on the market. Tools can be a valuable and useful addition to any software development life cycle but must be chosen carefully, checked out before commitment, and used in the proper perspective. This means that the tool should be recognized as a tool and the results should be evaluated carefully and action only taken on specific results generated by the tool.

6. Software Documentation

Perhaps the weakest link in modern software development has been documentation. There are a number of sources of information which provide guidelines concerning software documentation. MIL-S-52779A and the IEEE quality assurance standards both provide requirements for documentation. MIL-S-52779A calls for all documentation standards and programing conventions and practices to be used for all software to be referenced in the quality assurance plan. The IEEE standard calls for identification of the documentation governing the development and verification of the software and an explanation of how the documents are to be checked. It further calls for a number of specific documents. These include a software requirements specification (SRS), software design description (SDD), and software verification plan (SVP). FIPS PUB 38 provides extensive guidance concerning documentation of computer programs and ADP systems. Software documentation is an extremely

critical aspect of software development in that it is the means of communication which the designer uses with his colleagues, management and the technical authorities of the customer. Although it is widely recognized that good documentation practices should be maintained for all system projects and there are ample guidlines with which to proceed, documentation remains largely inadequate. Much of it is old, it is poorly written or written in such a manner as to be incomprehensible to the average reader, it may not be thorough or leave out elements which are critical to the software in question, or it hasn't been changed to reflect current practices. Software quality assurance must realize the requirement for good documentation and take steps to ensure that documentation which accompanies developed software is complete, clear, accurate and concise.

7. Configuration Management

Configuration management consists of identifying the configuration of the software system at discrete points in time. The purpose is to systematically monitor changes to this configuration and maintain the integrity and traceability of this configuration throughout the system life cycle. It is primarily concerned with ensuring the integrity and continuity of design. Quality assurance, through configuration management, should enforce the following: (a) Configuration Identification-A system of recording the technical description of individual computer programs and supporting documentation, thus documenting the functional and physical characteristics of the configured item. (b) Configuration Control-applies to configured software and documentation after they have been released. It also provides a control for changes and library features. (c) Configuration Status Accounting-the recording of the status of the system's configuration. The purpose is to

know exactly what the current configuration is, and how it was achieved.

III. A SURVEY OF MILITARY AND CIVILIAN SOFTWARE QUALITY ASSURANCE PROGRAMS

A. INTRODUCTION

As stated in [Ref. 22] the purpose of a quality assurance program, "Is to assure that software developed, acquired or otherwise provided under the contract complies with the requirements of the contract". Another definition of quality assurance is, "A planned and systematic pattern of all actions necessary to provide adequate confidence that the items will perform satisfactorily in actual operation" [Ref. 28]. Still another definition is, "The early detection and correction of deficiencies and the evaluation of overall quality performance" [Ref. 29]. Although pros and cons on the merits of each of these definitions can be thought of easily, they all have the same goal - prevent customer complaints.

Quality assurance achieves this goal through control. This control function is based on the existence of some type of plan and the control function is simply ensuring adherence to that plan. Effective control will detect deviation from the plan early, before it actually occurs. Ineffective control detects deviation as it happens, when it's too late. Two key factors in effective control are (1) total knowledge of the plan and (2) establishment of milestones against which progress on the plan can be measured. By monitoring these milestones, action can be initiated to prevent potential deviation from the plan.

In essence, quality assurance is not something that can be added later in the software development process. It is not the job of a single person or group of persons to see that quality is added at the right time and in the right

amount. Quality assurance begins at the start of the development process and is continually added at each step along the way. The primary job of the quality assurance group then becomes the development of the quality assurance plan and once it is developed, the management of it throughout the development process.

The next sections of this chapter will discuss standards which are in use for developing quality assurance plans. Following that will be a discussion of the typical software development cycle. The final section of the chapter will discuss the quality assurance programs in use at two major civilian companies and at the Naval Ocean Systems Center.

B. SOFTWARE QUALITY ASSURANCE PLAN STANDARDS

1. Military Specification 52779A

Mil-S-52779A applies to the acquisition of software either alone or as part of a complete system. It requires the establishment and implementation of a software quality assurance program by the contractor.

Paragraph 3.2 of Mil-S-52779A deals with the software quality assurance plan. According to this specification [Ref. 22] any software quality assurance plan will include the following areas:

1. Tools, Techniques and Methodologies: What are they and how will they support the overall Quality Assurance Program? Examples include: Operations Research - Systems Analysis, functional and performance requirements analysis, error analysis, software optimization tools, specification tracing and coding conventions.
2. Computer Program Design: How will design logic, fulfillment of requirements, completeness and compliance with specified standards be evaluated?
3. Work Certification: How will the description, authorization, and completion of work be certified or approved?
4. Documentation: What documentation standards and program conventions will be used?

5. Computer Program Library Controls: How will different computer program versions be identified and documented? The objective here is to insure that only approved modifications are made and implemented.

6. Reviews and Audits: How will reviews and audits be conducted to insure traceability from initial requirements to final product?

7. Configuration Management (CM): How are Software Quality Assurance and CM related?

8. Testing: This section includes the following areas -

   a) Analysis of requirements to determine testability

   b) Review of test requirements and criteria to insure adequacy, feasability, traceability and satisfaction of requirements.

   c) Review of test plans, procedures and specifications for compliance with contractual requirements and to insure all authorized changes are implemented.

   d) Verification of tests.

   e) Certification of test results.

   f) Review and certification of test reports.

   g) Maitenance of test material to insure repeatability.

   h) Support software and hardware used during development must be acceptable to the government.

2. IEEE Standard for Software Quality Assurance Plans

The purpose of this standard is to provide uniform, minimum acceptable requirements for preparation and content of Software Quality Assurance Plans. The standard applies to the development and maintenance of critical software (i.e. where failure could impact safety or cause large financial or social losses). For non-critical software or software already developed a subset of the requirements may be used [Ref. 23].

The following are the major sections and subsections of the plan as outlined in [Ref. 23].

1. Purpose.
2. Reference Documents.
3. Management.
   a) Organization

b) Tasks

c) Responsibilities

4. Documentation

a) Purpose

b) Minimum Required Documentation: Software Requirements Specification, Software Design Description, Software Verification Plan.

c) Other: Computer Program Development Plan, Configuration Management Plan, Standards and Procedures Manual.

5. Standards, Practices and Conventions.

a) Purpose

b) Content: Document Standards, Logic Structure Standards, Coding Standards, Commentary Standrds.

6. Reviews and Audits.

a) Purpose

b) Minimum Rquirements: Software Requirements Review, Preliminary Design Review, Critical Design Review, Functional Audit, Physical Audit, In-Process Audits.

7. Configuration Management.

8. Problem Reporting and Corrective Action.

9. Tools, Techniques and Methodologies.

10. Code Control.

11. Media Control.

12. Supplier Control.

If any of the above sections are not pertinent to the project for which the plan is being written, a statement stating this non-applicability should be included under the section heading along with reasons why it is not applicable. If additional sections are needed they may be included at the end.

C. PHASES OF THE SOFTWARE DEVELOPMENT PROCESS

It is generally agreed upon that the software development process consists of at least the following seven phases [Ref. 24] : Conceptual, Requirements Definition, Design,

Coding and Checkout, Testing, Integration, and Operational. Software definition takes place in the Conceptual phase. This consists of feasability studies, trade-off studies and analyses to define specific requirements to be allocated to computer resources. Once these requirements are defined and documented they form the basis for a draft system specification which will be used during the following phases.

During the second phase, Requirements Definition, it is determined which system requirements will be implemented by software. Through analysis it is determined which software functions are needed and the inputs, processing, and outputs that are required for each function. Also part of this phase is the finalization of the system specification and the preparation of the draft software requirements specification.

Following the Requirements Definition phase is the third phase of the development cycle, Design. The object of this phase is to come up with a software design that will implement the functions identified during the Requirements Definition phase. The design will include actual algorithms and equations along with control logic an data operations to be performed. The finalization of the software requirements specification and the preparation of the draft software design specification will also take place during this phase.

The fourth phase, Coding and Checkout, includes translating the software design into a computer programming language. Usually it is a high-order language but it may also be assembly language. Once compilation and assembly errors are corrected each individual program module is executed to remove obvious errors. This procedure constitutes the checkout.

Once coding is complete the fifth phase of the cycle, Testing, begins. Here the software which has been developed is tested to show that it is consistent with system and software requirements.

During Integration, hardware and software are brought together and system and operational testing is conducted. The object of the testing is to insure the satisfaction of system requirements in the actual or simulated environment.

The last phase of the cycle is the Operational phase. The software has been accepted for use and the only remaining activities are maintenance and modification. Figure 3.1 [Ref. 24] shows the software development process

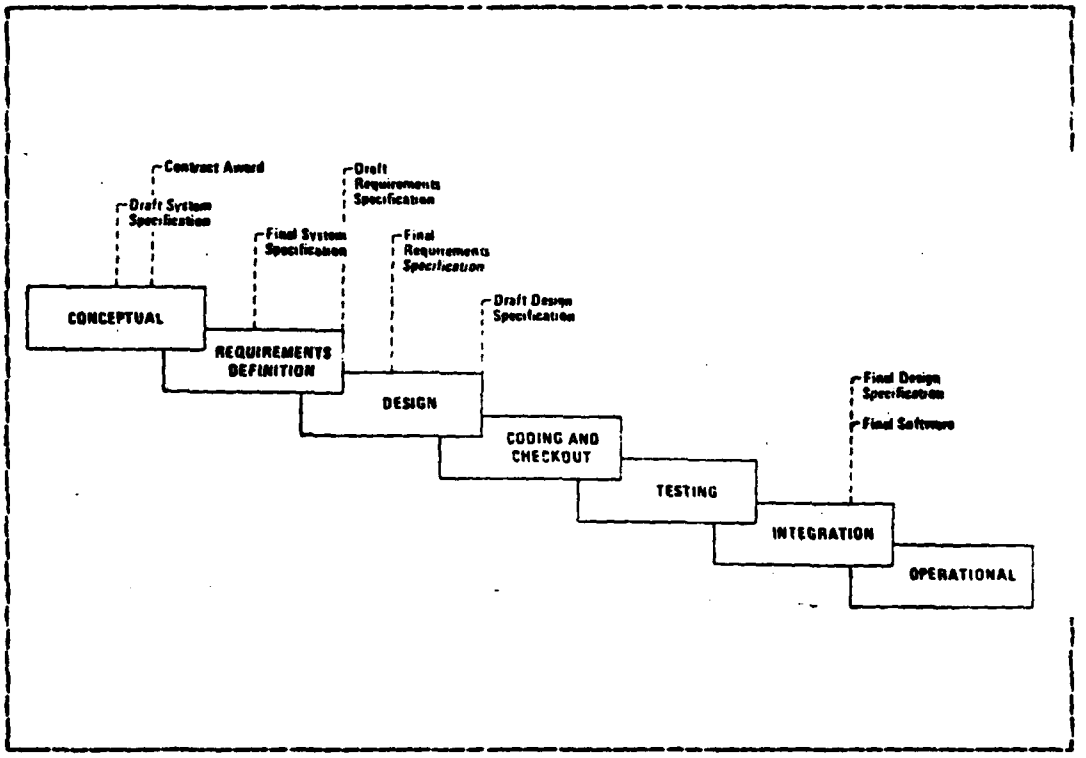


Figure 3.1 Software Development Process.

and the key outputs of the phases. Figure 3.2 [Ref. 24] shows how the quality assurance activities fit into the development process. Figure 3.3 [Ref. 29], although it does not specify all seven phases decribed above, it does show the activities of not only the Quality Assurance group but

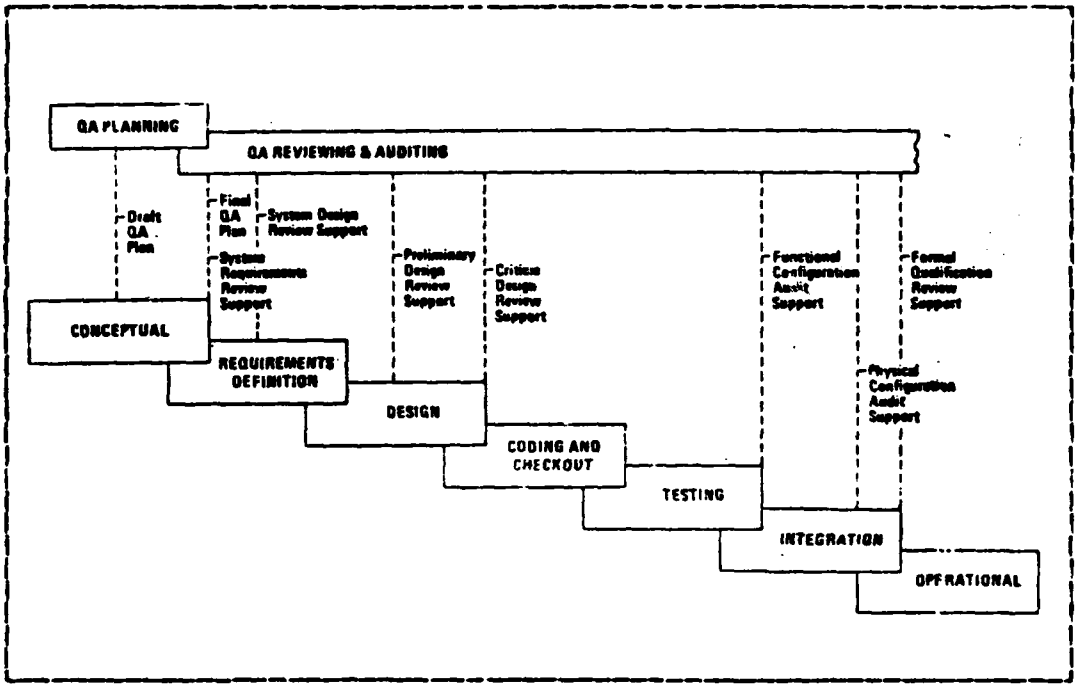


Figure 3.2 Software Quality Assurance Process.

of the groups in the project organization. The activities listed are all quality assurance related.

D. QUALITY ASSURANCE PROGRAMS

1. TRW Defense Systems Group

a. Background

Kurt F. Fischer [Ref. 30] writes:

> At TRW Defense and Space Systems group the need for a division wide organization to assist software management became quite apparent during the late 1960s. Like most other software vendors, TRW was concerned about the frequent cost overruns and schedule slippages in its software projects, and decided to develop methods to turn this trend around. As part of that decision, it established its first quality assurance staff in 1969.

THE PROJECT ORGANIZATIONS' ROLES IN THE SOFTWARE QUALITY ASSURANCE PROGRAM

DEVELOPMENT CYCLE PHASE	PROJECT ORGANIZATION				
	SYSTEM ENGINEERING	SOFTWARE DEVELOPMENT	TEST AND INTEGRATION	PROJECT MANAGEMENT	QUALITY ASSURANCE
Requirements Definition	• Requirements traceability • Technical review of specs	• Prepare Programming Standard Document (PSD) • Develop/implement Programmer Training Plan			• Prepare Project SQA Plan • Audit traceability analysis accomplishment • Audit spec review accomplishment • Review spec compliance to documentation standards • Review PSD scope for completeness • Review training plan for scope • Audit training attendance • Assure SRR action item closure
	• Prepare SW Req's Rev (SRR)			• Conduct SRR • Prepare Configuration Management Plan (CMP)	• Assure proper participation • Review for compliance with SQA plan
Preliminary Design	• Requirements traceability analysis • Technical review of specs and Interface Control Documents (ICD)				• Audit traceability analysis accomplishment • Audit spec and ICD review accomplishment • Review compliance to documentation standards
		• Complete PSD		• Establish engineering review/change control board	• Review PSD for completion • Audit implementation of standard
		• Initiate SW Development Folders (SDF)	• Trace test requirements to test plans		• Audit existence and maintenance of SDFs • Audit traceability analysis accomplishment
	• Technical Review of PDR material	• Prepare Prelim. Design Review (PDR)		• Conduct PDR	• Assure PDR action item closure
Detail Design and Code	• Technical review of specs and ICDs	• Trace requirements to design implementation			• Audit traceability analysis accomplishment • Audit spec and ICD review accomplishment
		• Conduct structured walk throughs			• Review compliance to documentation standards • Audit walkthrough accomplishment
	• Technical review of test plans • Tech. Rev. of CDR Mat'l	• Prepare Critical Design Review (CDR) • Maintain SDFs		• Conduct CDR	• Audit Test Plan review accomplishment • Assure CDR action item closure • Audit SDFs maintenance
Test and Operations			• Trace test plans to test procedures • Conduct test readiness meetings		• Audit traceability analysis accomplishment • Audit hookup accomplishment and procedures • Audit test pick review accomplishment
	• Technical review of test procedures	• Technical review of test procedures	• Conduct post-test meetings		• Review compliance to documentation stds • Certify test results and reports
		• Prepare DRs		• Prepare preship audit • ERB (CCB) DR disposition	• Conduct preship audit • Monitor expedite DR closure
	• Prepare Discrepancy Reports (DR)		• Prepare DRs		• Prepare statistical report on DR activity

Figure 3.3 Development Process for Quality Software.

At TRW the software quality assurance functions are performed by an organization titled Product Assurance. This organization is headed by a vice- president level staff director. Figure 3.4 [Ref. 30] illustrates the corporate

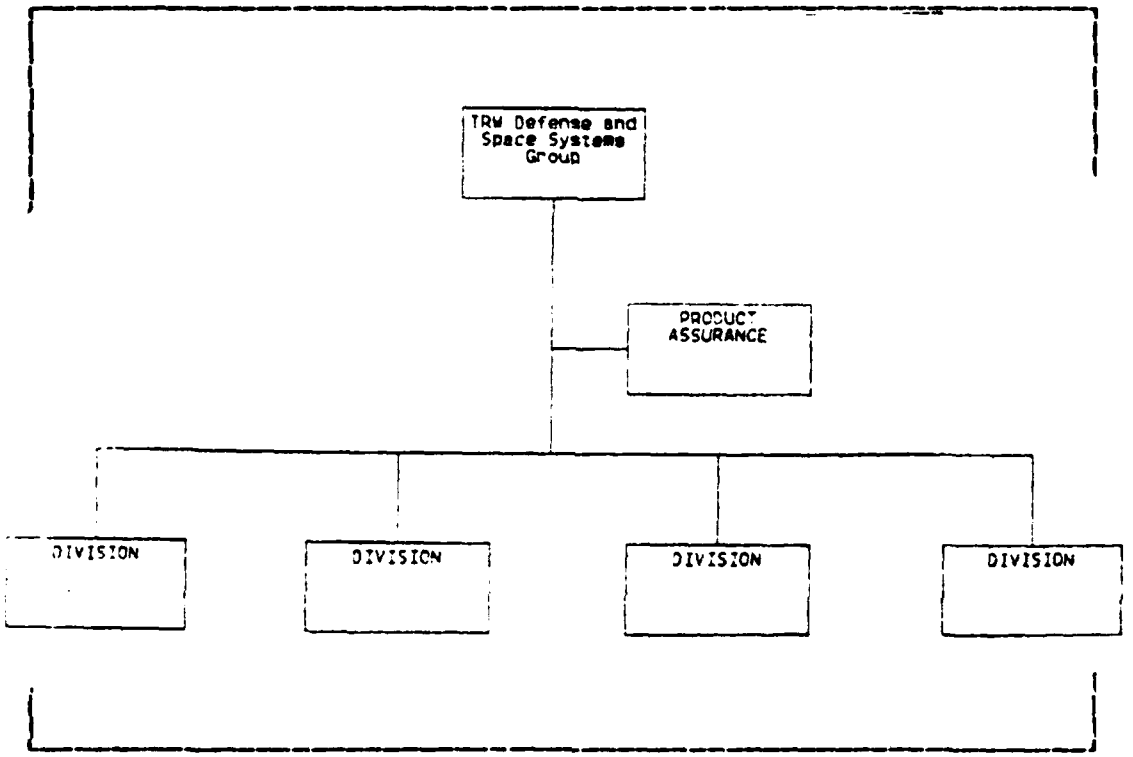


Figure 3.4 TRW Corporate Organizational Structure.

organizational structure. The Product Assurance organization actually performs a dual role: Quality Assurance and Configuration Management. The reason for this is that both areas have been found to share similar characteristics.

1. They perform staff oriented functions.
2. The performance of their functions is often times more credible when done by an independent organization.
3. Staff personnel share many aptitudinal characteristics (e.g. close attention to detail, preference for wide visibilty tasks) [Ref. 30].

Product Assurance on the product level is headed by an Assistant Program Manager for Product Assurance who has responsibility for both Quality Assurance and Configuration Management. The APM for Product Assurance receives his direction from the project manager, yet he and his staff remain organizationally independent from the project by reporting functionally to the Product Assurance organization at the corporate level. Figure 3.5 [Ref. 30] illustrates the project organizational structure.

b. Quality Assurance Objectives

To achieve the stated project quality assurance objectives, the following activities [Ref. 31] are performed by the Quality Assurance group at TRW:

1. Assure direct traceability between the subsystem specifications and the development specifications. In addition assure direct traceability between development specifications and the test plan and from there to test procedures. QA will additionally insure that all requirements are traceable to the product specification.
2. QA will develop and maintain a software requirements matrix. This matrix will be maintained throughout the software development cycle.
3. All documentation generated during the project will be reviewed by QA personnel.
4. Conduct audits of the software development process.
5. QA personnel will participate in all formal reviews and audits (e.g. Software Requirements Review, Preliminary Design Review and Critical Design Review).
6. Insure the implementation of built in checks and balances.
7. QA personnel will monitor and witness the Preliminary Qualification Test (PQT) and Formal Qualification Test (FQT). These test results will be cosigned by QA personnel. All QA test records will be maintained.
8. QA personnel will monitor the Configuration Management practices during software development. They will also test to insure the integrity of the software configuration.
9. QA personnel will participate in all Configuration Change Board (CCB) meetings and provide a review of all proposed changes in the software development and test process to again insure the integrity of the software configuration.

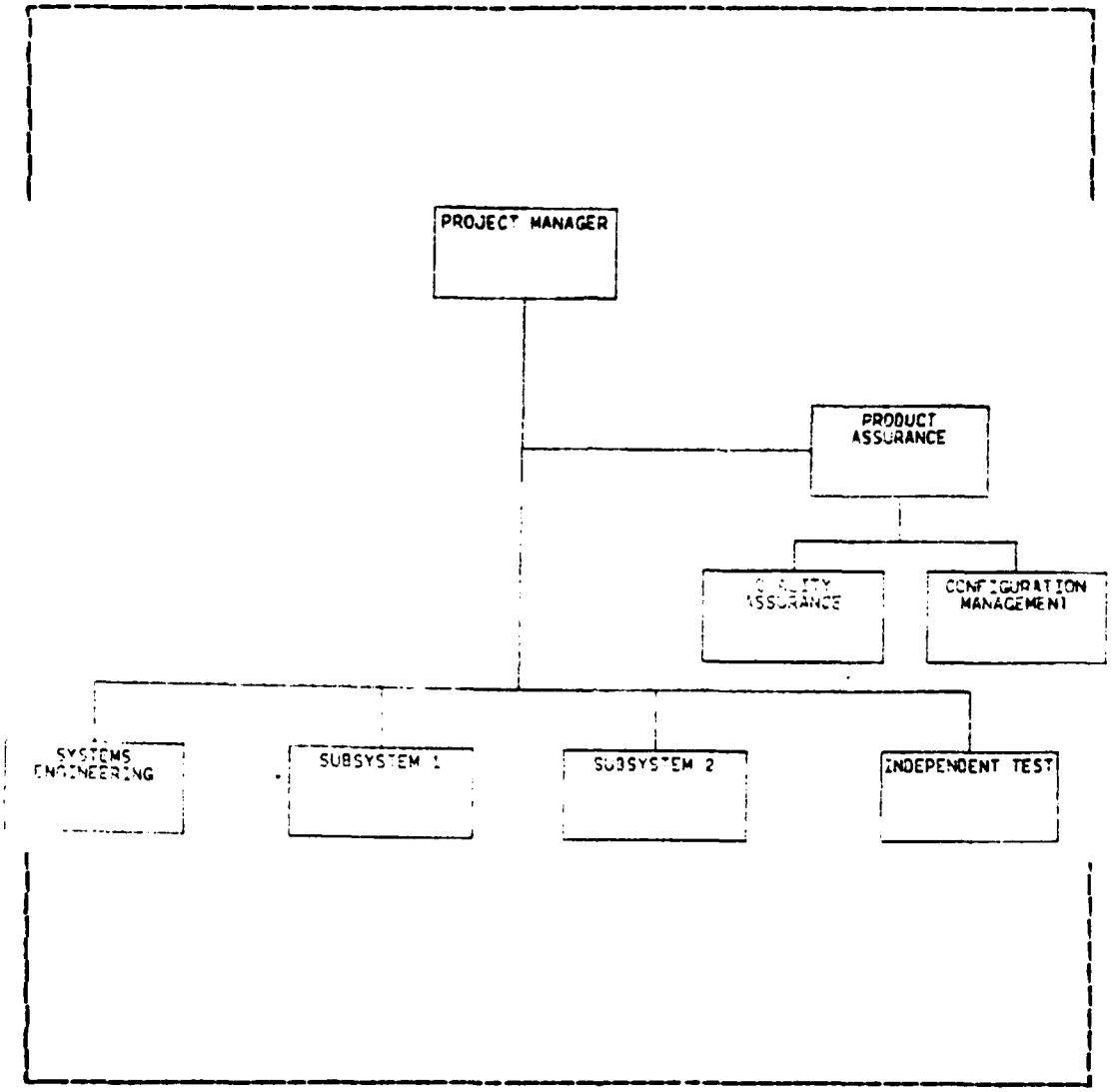


Figure 3.5 TRW Project Organizational Structure.

10. QA will support the development of project Software Standards and Procedures.
11. Verify that all requirements and functional capabilities have been satisfied by software testing.
12. QA personnel will insure that the delivered software package meets contractual requirements.
13. A system for tracking Software Problem Reports (SPR) and Design Problem Reports (DPR) throughout the software development and production life will be developed and maintained by QA personnel.

14. To insure the necessary implementation and support for all QA and CM activities, Quality Assurance will participate in the selection process of all software tools.

c. Quality Assurance Planning

The successful implementation of an effective Quality Assurance Program relies heavily on quality assurance planning during the early phases of software development. At TRW QA planning is accomplished by a complete review of the early project documentation. Examples of such documentation include the Contract Statement of Work, System Specifications and Project Plan among others. Once this initial review is completed the Quality Assurance Plan is prepared. This plan contains the functions, tasks, and responsibilities of the Quality Assurance group and also identifies the quality assurance tools needed to insure software quality in the areas of accountability, testability, usability, maintainability and reliability. Upon completion the plan is reviewed by other project organizations and approved by both the Program Manager and the customer. Upon approval task assignments are made to carry out the activities outlined in the previous subsection. As noted in [Ref. 30] these assignments are based on level of effort and must remain flexible to adapt to:

1. The needs of the current phase in the development life-cycle.
2. Shifts in attention needing areas (.eg. technical problems).
3. Unexpected demands placed on Quality Assurance by the Project Manager.

Once the QA Plan is approved the Quality Assurance polocies and procedures are written describing the methods and procedures to be used in the implementation of quality assurance requirements defined in the:

1. System Specification.
2. Contract Statement of Work.

3. Project Plan.

4. QA plan.

d. Software Standards

As Kurt F. Fischer points out in [Ref. 30], "The purpose of software standards is to improve the maintainability and readability of the software product". TRW has developed a comprehensive and detailed program to deal with software standards. According to [Ref. 30] this program has been successful for two reasons:

1. Software standards are not dictated from the executive offices or from Quality Assurance, but are developed out of close communication among the Design, Development, Test, QA and Project offices.

2. A tool has been provided to automatically check the software against most of the standards. This allows programmers to audit themselves so that there are no surprises at turnover time.

The Software Standards and Procedures (SSP) [Ref. 31] document contains the quality provisions, instructions and standards for each project. The SSP deals with standards concerning software, firmware, design, development and testing. The categories of standards included are:

1. Source code formatting standards

2. Techniques to be used in software/firmware design, code, test and update.

3. Standards dealing with QA tool development and their use during design development and checkout.

Waivers and deviations serve to complement the standards. They allow permanent or temporary relief from compliance with the standards due to technical difficulties, inefficiencies or schedule impact. All waivers or deviations must be approved by both the QA Manager and the Project Manager.

e. Quality Assurance Tools

The use of software tools should only be considered where it will prove to be more cost effective and more accurate to have the task automated rather than performing it manually. Tasks which may fall into this category are often tedious, menial, boring, difficult, error prone, repetitive and costly [Ref. 30].

Although TRW is currently using improved and updated tools the following are examples, taken from [Ref. 30], of the types of tools in use at TRW:

1. Product Assurance Confidence Evaluator (PACE) - PACE is designed to quantitatively evaluate how thoroughly and rigorously a program has been tested.
2. FORTRAN Code Auditor (FCA) - This tool audits coding standards and by doing so allows enforcement of these same standards.
3. Structured Programming Auditor (STRUCT) - STRUCT is used to ensure programs comply with the structured programming standard. It is executed after PACE because it relies on output from PACE. It evaluates program structure based on the following six constructs: sequence, if-then-else, do-while, do-until, case, escape-from-loop.
4. Units Consistency Analysis (UCA) - This is used on FORTRAN source code and the associated data base. It scans the code and interprets equations. By referencing the data base for the variables used in the equation it determines if the units in the assignment statement are in fact consistent.

f. Quality Assurance Reviews and Audits

Review and audit points are established during the QA planning phase to ensure that design, code, inspection, testing, and documentation are compatible.

As used at TRW reviews serve as quality assurance critiques of documents while audits are critiques of processes. Evaluation criteria for documents [Ref. 30] consists of the following:

1. Adherence to format and pagination.
2. Clarity of objectives.
3. Technical content.

4. Interdocument consistency.

5. Traceability to higher level specifications.

The audits conducted by Quality Assurance personnel [Ref. 30] serve the following four functions:

1. They assess compliance of source code and documentation to software standards and procedures.

2. They assure traceability of requirements.

3. They determine the satisfaction of system requirements during system test and acceptance phase.

4. They assess test sufficiency.

The following list contains the audits conducted by the quality assurance personnel at TRW. A brief description of each audit is also included.

1. Unit Development Folder (UDF) Audits - The UDF is a non-deliverable item which provides a mechanism for internal control and also provides management visibility for software development. The UDFs are prepared and maintained for each software unit provided by the project. The definition of a unit is a single routine or a group of logically related routines. The UDF is a collection of all requirements, design data, code, and test data pertaining to a specific unit. The UDF serves as the primary surveillance mechanism for the quality assurance personnel during the design, coding and unit test phases of the software development project. Each UDF is audited. This audit is divided into three phases in an effort to provide early detection and correction of possible errors. Each phase is designed to audit a specific area of the development process. Following is a description [Ref. 31] of each phase:

   a) Phase I - verifies appropriate UDFs have been initiated, proper cover sheets and inserts have been included, requirements have been stated in the requirements section and that the design section contains the current working design.

   b) Phase II - A desk check and automatic code audit is performed to determine if code isbeing produced in accordance with established project standards and guidelines.

   c) Phase III - Verifies that each UDF audited contains a compilation of test results and analyses necessary to demonstrate that the unit of code has been debugged, init development testing is complete and that an up-to-data design description exists.

2. Test Data Folder (TDF) - The TDF is the primary working document for the test group. As such it also provides a surveillance vehicle for quality assurance personnel. During the integration and acceptance testing phases of the development project. The TDF contains the test requirements, test plan, test procedures, test execution reports and software

problem reports. Quality Assurance reviews the folder to insure adequacy, completeness and conformance to standards outlined in the Software Standards and Procedures (SSP).

3. Configuration Management Audits - Quality Assurance uses this audit to monitor the configuration management activities to insure that they comply with both the CM plan and documented procedures.

4. Interface Verification Audit - This audit is conducted as early as possible in an effort to identify and correct possible interface problems. QA personnel examine the requirements design and program specifications.

5. Preliminary and Detailed Design Audits - Conducted prior to the Preliminary Design Review (PDR) and the Critical Design Review (CDR) respectively, these audits are concerned with the format and content of design documentation and project test plans. The results of these audits are then discussed at the PDR and CDR respectively.

6. Independent Quality Audit - Not only do the Product Assurance personnel audit the various areas of the software development project, but they in turn are audited by the corporate Product Assurance organization. This audit does in fact cover the whole project, however, the QA and CM areas are examined in detail. Upon completion of the audit both the Project Manager and Assistant Project Manager for Product Assurance receive copies of the audit report. They also receive any Corrective Action Requests (CAR) which document any discrepancies found by the audit.

7. Audit Reports - The findings/results of the above audits are provided to both the Project Manager and the Assistant Project Manager for the appropriate development area. In addition to the findings recommendations are also included. A periodic summary which details the number/type of audits performed, the discrepancies found, all corrective action in progress or implemented and a follow-up on corrective action on-going from previous reports is also prepared by Quality Assurance and provided to the Project Manager.

As previously noted at the beginning of this subsection reviews are conducted to critique documentation. What follows is a list of the reviews conducted at TRW. A brief description of each review is also provided.

1. Software Requirements Review - This review is conducted upon completion of the Software Requirements Definition phase. QA personnel sit as members of the review board. The purpose of the review is to insure that the software requirements specifications for the proposed software project do in fact match the users operational requirements for the system.

2. Preliminary Design Review - In addition to conducting the Preliminary Design Audit, Quality Assurance personnel act as recording secretary during this

review. The PDR [Ref. 31] reviews each development specification for the following:

a) Traceability of requirements specification to the development specification.

b) Requiremntes satisfaction, interface definition and specification content.

3. Critical Design Review - In addition to performing the Detailed Design Audit Quality Assurance personnel serve as recording secretary. The emphasis of the CDR is on the preparation of required materials, briefing content and the allocation of each requirement to a functional design element [Ref. 31].

4. Design Walkthroughs- These walkthroughs are held early and oftenduring the Design phase. The walkthroughs are conducted by technical personnel who are taken through the design on a step-by-step basis informally by the designer. This done in an effort to monitor the consistency of the design approach, satisfaction of requirements and completeness [Ref. 31].

g. Testing

In [Ref. 31] it states that Quality Assurance has review authority for all test plans and procedures initiated by TRW for either formal or informal testing. QA performs a selective review of documentation to insure that test cases and test procedures directly correlate with the software requirements. The two primary functions performed by Quality Assurance durig the Testing phase are test auditing and the inspection and surveillance of formal tests. Both these areas are described below.

1. Test Audit - This audit is conducted at the end of each test phase and its primary purpose is to:

a) Assure that software configuration management procedures are being followed.

b) assure that the test specifications being used by the test group are the current approved versions.

c) Assure that test reports identify proper test procedures and software configuration; specify the test analysis, and if any deficiencies were noted how they were explained and accounted for.

d) Verify that test procedures provide a step-by-step rationale for conducting a test and that test results comply with acceptable criteria specified in the test procedure.

e) Verify that Test Data Folders comply with approved formats.

f) Verify compliance by the test team with stated management procedures for change control, discrepancy reporting, and test reporting [Ref. 30].

2. Inspection and Surveillance of Formal Tests - This is an on-site activity performed by QA personnel during program testing. The purpose is to:

   a) Monitor all tests to ensure that the actual tests performed are those specified in the documented test procedures.

   b) Assure all potential discrepancies are recorded in the approved manner.

   c) Compare configuration of hardware/software components used in the test against the configuration identified in the etst procedures.

   d) Certify that analysis of test results is correct, the test satisfies the intended requirements and acceptable criteria and the Test Data Folder is complete.

   e) Assure that master copies of test procedures, test results and test reports are maintained and available from a centralized records center [Ref. 30].

3. Test Changes - The test director may make corrections of typographical errors in the test parameters as long as they do not deviate from specified requirements, he may also make changes in the test set-up and in addition he may make changes in the procedure step sequence provided that test parameters or tolerance accuracies are not changed. All changes must be documented and will be reviewed for approval by the Test Review Board and Configuration Control Board. Any changes causing a deviation from specified requirements must be submitted through both the Configuration Control Board and the customer [Ref. 31].

4. Test Change Request (TCR) - This is the vehicle used for requesting changes to test procedures. A description of the problem and the proposed changes are included on the TCR. Upon approval by the CCB it provides both the solution to the problem and the means for implementing that solution. Figure 3.6 [Ref. 32] is a sample TCR used at TRW.

h. Problem Reporting and Review

In addition to the problem reporting and review procedures, the procedures used at TRW for the control of changes to the software product will also be discussed. The change control procedures fit in at this point because more often then not changes are made in response to problems which arise during the development process.

1. Configuration Control Board (CCB) - At TRW the Configuration Control Board has been established to review and approve changes to documentation and

TCR No. _____

TEST CHANGE REQUEST

PAGE OF

ORIGINATOR	SITE	MAILING ADDRESS	PHONE	DATE

DOCUMENT TITLE	NUMBER	REV	DATE ISSUED

TCR TYPE: ☐ ROUTINE ☐ PRIORITY

PROBLEM

PROPOSED CHANGE

APM SIGNATURE / DATE	PA SIGNATURE / DATE

CMO DATE RECEIVED	TRB INITIAL REVIEW DATE	TRB ACTION ASSIGNED / DUE DATE

CLASSIFICATION: ☐ I ☐ II

RELATED SPR/DPR:

ACCEPTED CHANGE

TESTS TO BE RERUN

DISPOSITION
- ☐ APPROVED FOR IMPLEMENTATION
- ☐ SUBMIT ECP
- ☐ REJECT AND CLOSE

SCHEDULED IMPLEMENTATION DATE

APM SIGNATURE / DATE	TRB CHAIRMAN SIGNATURE / DATE

Figure 3.6 Test Change Request.

software. Regularly scheduled meetings are held but if circumstances warrant it the Project Manager can call a special meeting. The project manager acts as chairman and members include personnel from Product Assurance, Data Processing Systems Engineering, Applications Software, Systems and Support Software, Data Processing Hardware, and Integration and Testing. The user may also take part in the meetings and in any event will be supplied a copy of the minutes.

The following are the documents used at TRW in an effort to maintain configuration control over a software development project. These documents are submitted to the CCB for its review and approval.

1. Design Problem Report (DPR) - This report is used to request changes to baselined (already approved) documents and also to initiate changes to formally baselined customer controlled documents. The DPR contains both a description of the problem and a proposed solution. Once it has been approved by the Configuration Control Board it provides the accepted solution to be implemented. Marked up change pages must be attached to the DPR to illustrate the changes being made. QA personnel monitor the resolution of all DPRs [Ref. 32]. Figure 3.7 [Ref. 32] is an example of the DPR used at TRW.

2. Software Problem Report (SPR) - The SPR is a request for permanent changes to internally controlled code. It includes a description of the problem, identifies the library and routines affected and proposes a problem solution. Once approved by the CCB it serves as an authorization to update the master library [Ref. 32]. Figure 3.8 [Ref. 32] is an example of TRW's Software Problem Report.

3. Temporary Modification Notice (TMN) - The TMN requests and implements temporary changes to baselined code. Included with the TMN are a listing of actual changes, reasons for the change, any restrictions, testing, verification and files affected. The TMNs are correlated to SPRs which implement the permanent change [Ref. 32]. Figure 3.9 [Ref. 32] is an example of a Temporary Modification Notice.

i. Benefits

Kurt F. Fischer notes in [Ref. 30] that TRW received the following benefits through the implementation of its QA Program:

1. It has provided increased management visibilty into the development process through reviews and audits.

2. Project risk has been reduced through better requirements traceability and more disciplined and thorough testing.

3. It has enforced software standards.

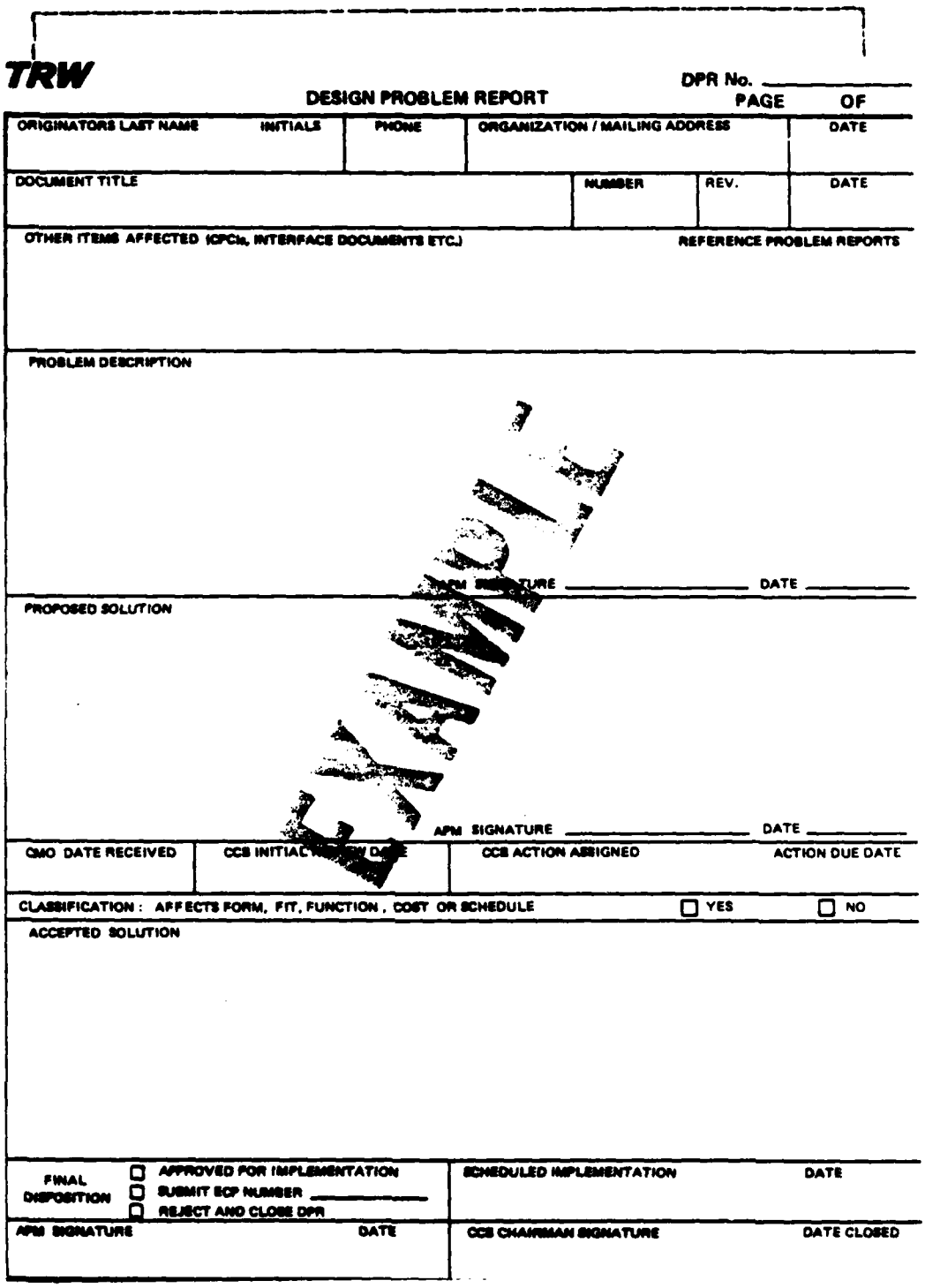


Figure 3.7 Design Problem Report.

4. The development and maintenance of software tools has been centralized.

5. Quality Assurance records have been centralized. These include, problem reports, deviations and waivers, reviews and audits, and test and inspection reports among others.

6. A skill center for personnel with multi-project visibility who are better able to prepare project plans and procedures.

7. An independent group assuring that deliverable items meet contractual requirements.

j. Lessons Learned

Kurt Fischer states, "Probably the largest lesson learned is that one key to the successful development of software is the employment of a strong QA activity", [Ref. 30]. In addition to the above, Fischer points out in [Ref. 30] that the following lessons were learned by TRW during the implementation of its QA Program:

1. Insure adequate QA participation during the proposal and contract definition phases.

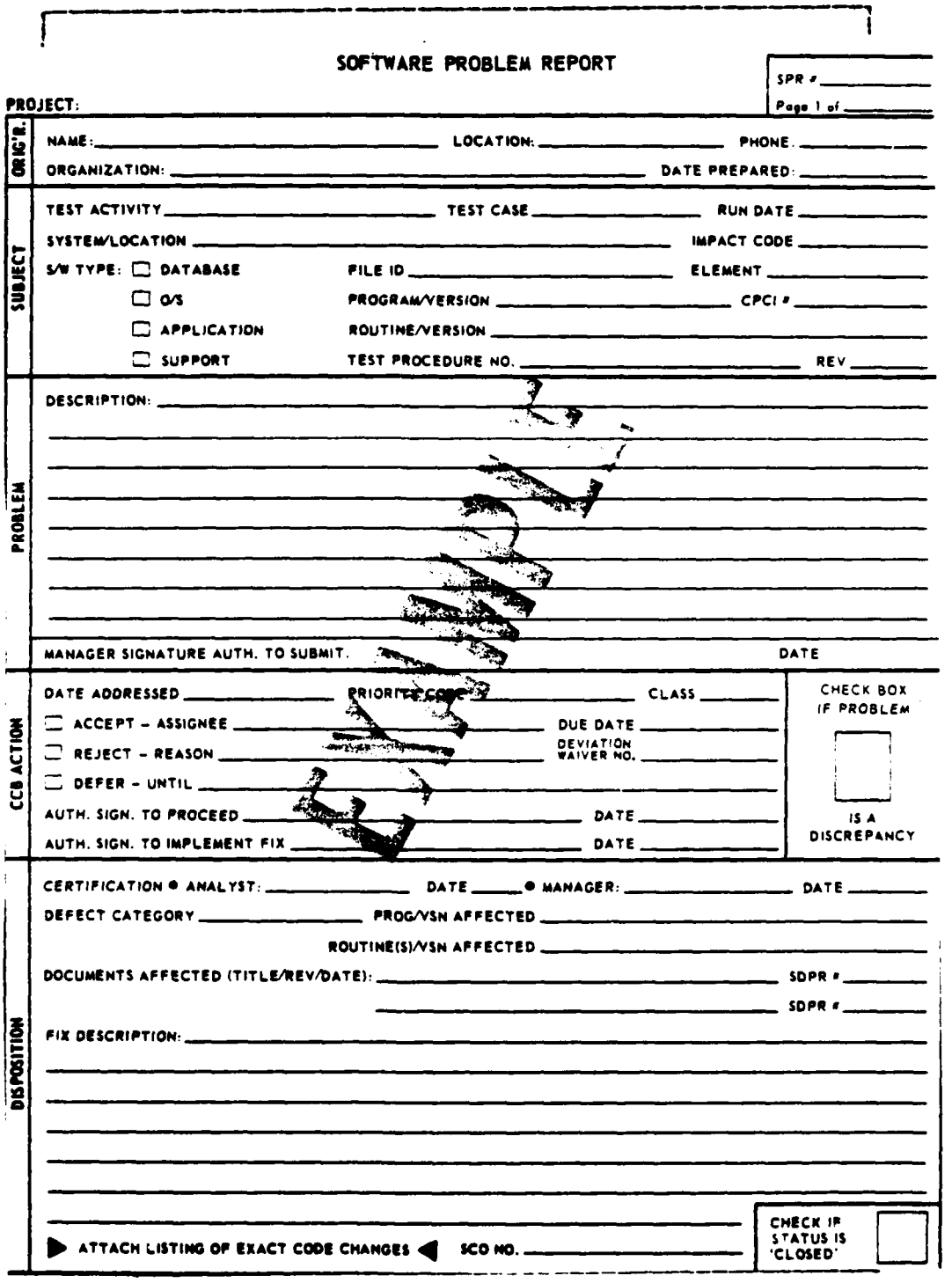
2. Hiring personnel knowledgeable in software and then training them in quality assurance is easier than hiring QA people and training them in software.

3. Perform the first audit early in the development processto allow plenty of time for corrective action.

4. Announce the audit well before it takes place. The object is to ensure it was done right, not to find problems.

5. Construct an audit checklist and distribute it to the area being audited at the same time the audit is announced. This eliminates subjective assessments by the auditor and informs the party being audited of the exact scope and depth of the audit.

6. Assign QA engineers on a long term basis so that they may develop a relationship with each project sub-group. QA engineers should colocate with the development personnel so that they might better understand the problems of other project members.



SOFTWARE PROBLEM REPORT

SPR # _____
Page 1 of _____

PROJECT:

ORIG'R.

NAME: _____ LOCATION: _____ PHONE: _____

ORGANIZATION: _____ DATE PREPARED: _____

SUBJECT

TEST ACTIVITY _____ TEST CASE _____ RUN DATE _____

SYSTEM/LOCATION _____ IMPACT CODE _____

S/W TYPE: ☐ DATABASE FILE ID _____ ELEMENT _____

☐ O/S PROGRAM/VERSION _____ CPCI # _____

☐ APPLICATION ROUTINE/VERSION _____

☐ SUPPORT TEST PROCEDURE NO. _____ REV _____

PROBLEM

DESCRIPTION: _____

MANAGER SIGNATURE AUTH. TO SUBMIT. DATE

CCB ACTION

DATE ADDRESSED _____ PRIORITY CODE _____ CLASS _____

☐ ACCEPT – ASSIGNEE _____ DUE DATE _____

☐ REJECT – REASON _____ DEVIATION WAIVER NO. _____

☐ DEFER – UNTIL _____

AUTH. SIGN. TO PROCEED _____ DATE _____

AUTH. SIGN. TO IMPLEMENT FIX _____ DATE _____

CHECK BOX IF PROBLEM ☐ IS A DISCREPANCY

DISPOSITION

CERTIFICATION ● ANALYST: _____ DATE _____ ● MANAGER: _____ DATE _____

DEFECT CATEGORY _____ PROG/VSN AFFECTED _____

ROUTINE(S)/VSN AFFECTED _____

DOCUMENTS AFFECTED (TITLE/REV/DATE): _____ SDPR # _____

_____ SDPR # _____

FIX DESCRIPTION: _____

▶ ATTACH LISTING OF EXACT CODE CHANGES ◀ SCO NO. _____

CHECK IF STATUS IS 'CLOSED' ☐

Figure 3.8 Software Problem Report.

2. Naval Ocean Systems Center

a. Background

The Naval Ocean Systems Center (NOSC) established its software Quality Assurance program to assist procuring activities in acquiring quality software. Quality software is defined as software which meets all requirements of operability, reliability, and maintainability. The expressed mission of the Software Quality Control organization is to provide assistance to project managers in the acquisition/management of higher quality software products through the implementation of certain standard practices. It provides a manageable structure to the software development process through document inspection, configuration management, and testing support. Standard techniques include inspecting and evaluating the documentation of computer systems, evaluating a projects configuration management procedures in regards to software documentation and computer programs, assuring the integrity of tested programs through program library control, increase user confidence through testing, and being an active participant on project Change Control Boards.

The activities of the Software Quality Control organization are geared to the projects life-cycle. This is true whether the project is short, requiring a minimum effort, or long, extending over a multi-year period. Because of this fact each SQC step is tied to the project plan, milestone schedule, and list of configuration items expected to be baselined [Ref. 33].

At NCSC the Quality Assurance office has been established at the directorate level. Figure 3.10 [Ref. 33] shows the NOSC organizational structure.

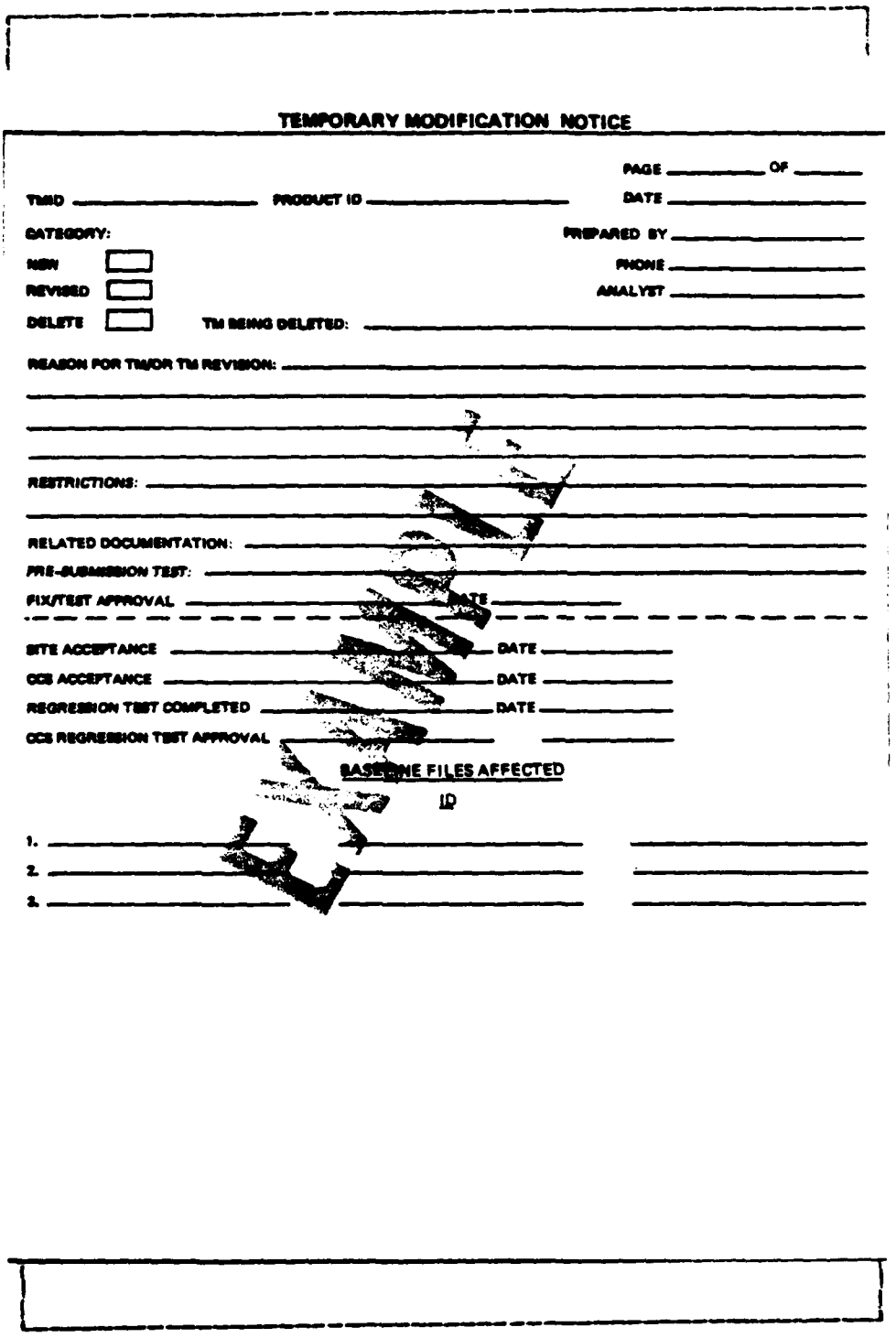


Figure 3.9 Temporary Modification Notice.

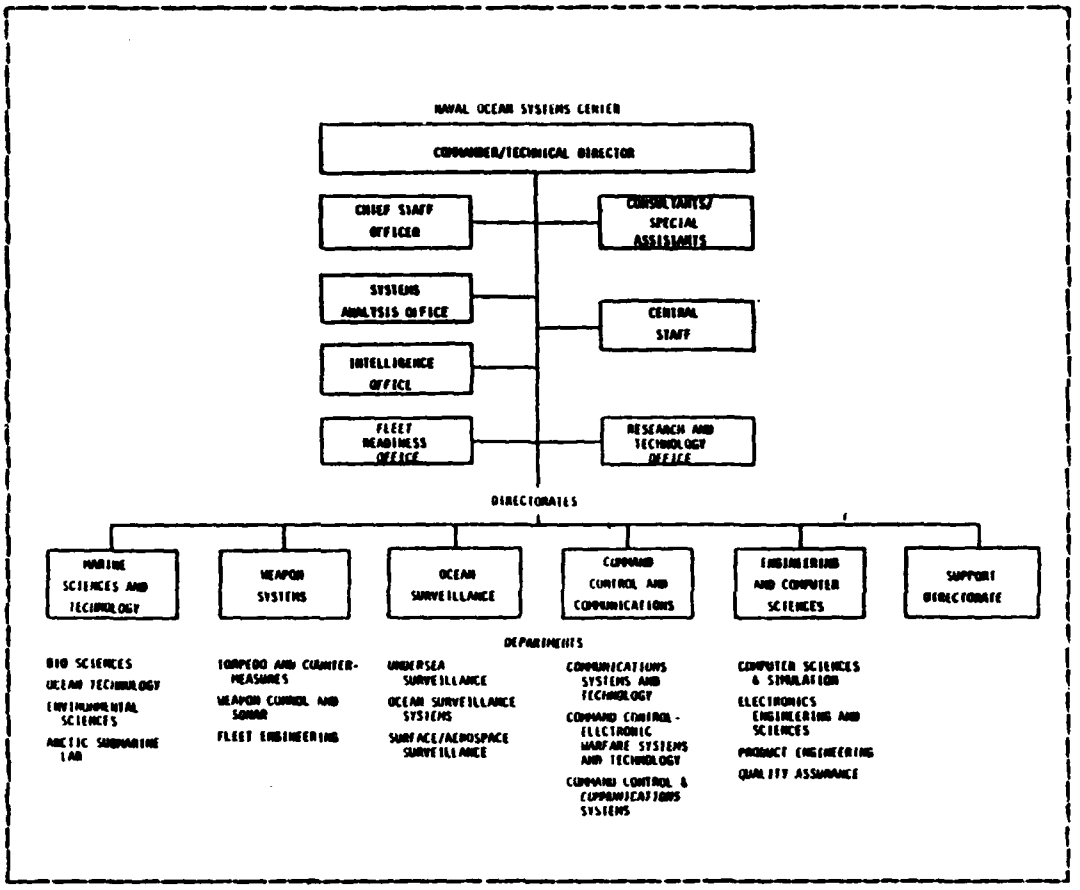


Figure 3.10 NOSC Organizational Structure.

b. Quality Assurance Objectives and Policies

The following are the most significant of the objectives established for the Software Quality Assuranc organization:

1. Ensure consistency of software baseline development.
2. Ensure compliance with design standards.
3. Ensure compliance with programming standards.
4. Ensure adequacy and completeness of testing.
5. Review all test results. [Ref. 33]

The following policies have been established at NOSC in regards to Software Quality Assurance:

1. The developing directorate has the basic responsibility for the quality of products delivered by NOSC. Each Director shall utilize the established quality assurance resources, as appropriate, to assure adequate quality of all end products.

2. The Director of the Engineering and Computer Sciences directorate acts as Center managements agent for product quality assurance on all Center projects. As such he will keep Center management informed of the results of reviews and audits of products developed and produced by the center.

3. The Quality Assurance office, which reports directly to the Director of the Engineering and Computer Sciences directorate, will be the point of contact for the coordination of all Center quality assurance activities. The QA office is responsible for keeping the Center's QA policies current and responsive to higher Navy directives and instructions. [Ref. 33]

c. Quality Assurance Planning

The Software Quality Control Plan is prepared by SQC personnel through interviews conducted with the Project Manager. The plan is coordinated with project plans and provides a description of how the elements of quality management will be applied to the project. Each SQC plan is tailored to project requirements.

For the Quality Assurance program to be effective the SQC plan must contain certain elements. These are: planning for products at the end of each task using management audits and reviews as a measure of completion of the products, developing documentation in the proper sequence, and accepting SQC inspection assistance to the Project Manager as an aid in ensuring successful system turnover [Ref. 33].

d. Software Standards

Software Quality Control personnel are responsible for developing and enforcing design and programming standards. In the area of design, standards dealing with

clarity, detail, logical efficiency, technical maturity, and consistency with functional specifications must be enforced.

Programming standards provide the required consistency in the technique and processing required for continued software support throughout the system's life-cycle. The programming standards deal with the following areas: logic and coding conventions, flow chart standards, intermodule communications, programming language structure and use, data design, module segmentation, and logic error checking.

Quality Assurance personnel review the programming effort with regard for compliance with standards. If any non-compliance is found they will take steps to ensure conformity with the standards.

e. Quality Assurance Reviews and Audits

The software review process exists so that a qualified decision may be made to recommend advancement from one phase to the next. Audits on the other hand are conducted to verify configuration items conform to specifications and other contract requirements. The results of reviews and audits are reported directly to the development directorate by quality assurance personnel.

In addition to the reviews and audits the process of baselining will be discussed. It is included here because it is an integral part of the audit and review process.

Baselines are a configuration management technique used to control the development of a software product. as stated in [Ref. 34],

> A projects software configuration is the prevailing state of its softwar e components. Those components which are subjected to systematic management are termed Configuration Items (CIs). Software CIs are qualified as being elements of an evolving software product which are set forth in technical documentation (including

specifications, drawings, and listings), and achieved in the computer program itself (resident on card, tape, or disk).

In [Ref. 33] it goes on to point out,

> Baselines are employed throughout the life-cycle of a configuration item to ensure orderly transition from one major commitment point to the next in the system engineering, software development production, and logistic support processes. Baselines are established at those points in a program where it is necessary to define formal departure points for control of future changes in performance, design, development, production, and related technical requirements.

The baseline is created upon acceptance of a development document or product.

There are four types of baselines used at NOSC, Functional, Allocated, Product, and Operational [Ref. 34]. A brief description of each follows.

1. Functional Baseline - This considered the highest level baseline. The technical documentation at this level delineates all the necessary functional characteristics, the tests which will be required to demonstrate their achievement, all necessary interfaces, and any and all design constraints. This baseline generally covers all the documentation produced prior to development of the software design and the formal System Design Review.

2. Allocated Baseline - At this the next lower level baseline, the performance oriented specifications, which are subordinate to the Configuration Items of the functional level, expand upon allocated functional characteristics. All documentation produced short of the Preliminary Design Review is covered by this baseline.

3. Product Baseline - This is considered the lowest level. The documentation at this point, which is subordinate to both the functional and allocated levels, defines the production, operation, maintenance, and logistic support phases of its life-cycle. This normally covers all documentation and programs produced prior to the Formal Qualification Review.

4. Operational Baseline - Once the developed system passes the Formal Qualification Review and has proved it meets operational requirements, the operational baseline is established. All modifications required to the system during its life-cycle will be performed from this baseline.

The baselining technique is also used by TRW during its software development process.

Following is a list of the reviews conducted by the Software Quality Control organization at NOSC. A brief description of each review is included.

1. Initiation Review - The purpose of this review is to affirm the Operational Requirement as the basic guideline for the project. Prior to the review the Operational Requirement should have been read by all members of the project team including software quality assurance personnel. The Operational Requirement is reviewed to ensure no requirements or constraints have been omitted [Ref. 33].

2. Systems Requirement Review - The objective of this review is to determine the adequacy of the developer's efforts in defining system requirements. The review is conducted once a significant portion of the system functional requirements have been established [Ref. 33].

3. Document Review and Substantiation - This occurs in each phase of the development life-cycle. The primary concern is to review draft documentation for completeness and correctness. Quality assurance personnel inspect and substantiate the contents of all documentation. The documentation under review is compared against established standards to ensure it contains the proper level of detail and that all the required content is present. Not only is a single piece of documentation reviewed by itself but it is compared to all associated documentation to ensure completeness and consistency. All deviations from standards and any technical problems, are noted and submitted to both the Project Manager and the developer for correction. Once the Operational Requirement has been reviewed and approved, the remainder of the documentation produced by the project is reviewed to ensure it is consistent with and meets the requirements set forth in the Operational Requirement [Ref. 33].

4. System Design Review - Once the project team has determined that the software requirements documents fulfill all requirements and present a suitable allocation of performance requirements between hardware, software, and human actions, the System Design Review is held. Also the identification, correlation, completeness, and risk of the software requirements documents is evaluated. Upon approval these documents are considered baselined [Ref. 33].

5. Preliminary Design Review (PDR) - The primary concerns of the PDR are the software design and the completion of requirements set forth in previously baselined documents. The items which are considered from the program design documents include:

   a) Computer program functional flow charts.

   b) Storage allocation charts.

   c) Control functional descriptions

d) Structure and organization of the data base.

e) Functional interfaces. [Ref. 33]

6. Critical Design Reviews - These reviews are conducted as individual system programs or modules are specified. The primary concerns here are that required standards are met, all prior baselined functions are fulfilled, and that prior to coding, the lowest level of design detail has been reached. The following items [Ref. 33] are under consideration from the program specifications:

   a) Compatability of design with functional interfaces.

   b) Data base interactions.

   c) Design integrity of logic diagrams, algorithms, storage allocations, and flow charts.

   d) Hardware interfaces.

   e) Human interfaces.

7. Formal Qualification Review - This is conducted upon completion of both the Functional Configuration Audit and the Physical Configuration Audit and after all discrepancies uncovered by these audits are corrected. The purpose here is to ensure that the system as developed and tested meets all requirements set forth in the Operational Requirements. From this review it is determined that:

   a) User requirements have been satisfied.

   b) Documentation is sufficient to support the system throughout its life-cycle.

   c) User functions are adequately described and documented.

   d) Testing is sufficient to ensure user confidence.

   e) All outstanding deficiency reports have been resolved.

   f) The outcome of the review will either be a successful completion or the determination that further development is necessary [Ref. 33].

8. Post Development Reviews - These reviews are conducted as the operational evaluations of the system are made. The focus is on system development problem areas, operational difficulties, and undetected deficiencies [Ref. 33].

The following list contains the major audits conducted by the NOSC Software Quality Assurance group during the software development process. As was the case with the reviews, a brief description of each audit is included.

1. In-Process Configuration Audit - This serves as the primary means of validating that development of a software Configuration Item has been completed satisfactorily. Once a document has gone through the review and substantiation process (described earlier) a record of the review forms and disposition of deficiencies is maintained. The updated document and all deficiency reports are inputs to this audit [Ref. 33].

2. Functional Configuration Audit - This is a critical comparison of an item's test/analysis data and functional specifications to validate that the item as designed and developed, meets all the functional performance requirements specified in its development specification [Ref. 35].

3. Physical Configuration Audit - This is a comparison of the "as built" item with its approved and released technical documentation. The objective is to ensure that the documentation is complete and is appropriate for operational maintenance and support purposes [Ref. 35].

Both the Functional Configuration Audit and the Physical Configuration Audit are conducted prior to the submission of a configuration item for formal acceptance. From [Ref. 33] the purpose of these audits is:

1. Confirm compliance to change control procedures and to ensure only approved changes have been implemented.

2. To ensure that objectives are sufficient.

3. To ensure the developed software product is the same as the specified software product.

f. Testing

Software test and evaluation, as conducted by the Software Quality Assurance personnel, is designed to ensure that the software, as developed, meets the original requirements set forth by the user/sponsor in the Operational Requirement and that it performs as defined in the system documentation. By conducting an analysis, technical evaluation, and a detailed review of project documentation the necessary inputs to prepare test plans, test specifications, and test procedures, are obtained. These documents are then used in the actual testing and evaluation of test results to verify that the system, as developed meets all technical specifications [Ref. 33].

Following is a list of the various test elements and activities. A brief discussion of each is included.

1. Test Plan - This is written by Software Quality Control personnel to define the scope of tests required to assure that the software meets all required specifications. Although the test plan is a high level document, from which the test specifications are written, it still must identify the degree of testing and the specific functions to be tested. The schedule for individual tests and a summary of the environment to be used are also included in the test plan. The plan is reviewed by the software developers and approved by the Program/Project Manager [Ref. 36].

2. Test Specifications - Software Quality Control prepares a test specification for each test in the test plan. Based on requirements set forth in the design documentation of the developing system, the test specification defines the basic test criteria and the general methods to be used in a specific test. Once a test specification is prepared it forms the basis for the development of test procedures. In addition to defining the scope of the specific tests, test specifications state the purpose of the test and identify the software, hardware, and/or system to be tested. There must be sufficient information in a test specification so that test procedures may be developed and so the results of defined tests may be evaluated. These are also reviewed by the software developers and approved by the Program/Project Manager [Ref. 36].

3. Test Procedures - These procedures are developed by the Software Quality Control personnel using test specifications, users manuals, and other relevant design documentation. The prime purpose of the test procedures is to present detailed instructions for both test execution and the evaluation of test results. The organization and structure of the processes are expressed in general terms in the test procedure along with constraints or assumptions imposed on their usage. A description of the total equipment, manpower, computer program, and supporting documentation required for operation is also provided. All hardware or software revisions or modifications must be specified along with any required pre-test checkout required to ensure a valid test environment [Ref. 36].

4. Software Testing - Software Quality Control personnel may perform testing in either of two environments: simulated or "on-site". In a simulated environment certain subsystems, links, and peripherals are available for the express purpose of production and testing. "On-site" testing is done using the system in real-life environment. The policies and procedures are set forth in the test plan, specifications, and proceduresused in accomplishing software testing [Ref. 36].

5. Test Reports - As each test is completed a Test Report is written to document the satisfactory or unsatisfactory completion of the test. Any and all deviations from test procedures or equipment malfunctions must also appear on the Test Report. Each apparent system discrepancy will be noted by the

submission of a seperate incident report. The Test Report will reference all completed test procedure steps to allow programmers to duplicate the conditions in which any apparent incident was discovered. All completed Test Reports become a part of the permanent system documentation [Ref. 36].

g. Problem Reporting and Reviews

As was the case with TRW the area of change control is also discussed in this subsection. The Configuration Control Board and the documents used in configuration control are discussed in the list that follows.

1. Configuration Change Control - The purpose of configuration change control is to manage and monitor changes or modifications to baselined configuration items. The sponsor, user, software developer, or any other member of the project organization may propose changes to the software. These, along with any problems uncovered during test and evaluation, are presented to the Configuration Control Board (CCB) as either Engineering Change Proposals, deviations, or waivers. The CCB in turn investigates all change requests, deviations, and waivers and based on documented analysis, recommends proposed action [Ref. 33]. The CCB is made up of representatives from all project affected activities. The Project Manager is the chairman and the Quality Assurance representative acts as recording secretary. Although the final decision regarding all changes ultimately rests with the chairman, he solicits expert advice from project participants such as Software Development, Systems Engineering, Quality Control, Logistics Support, Fleet User, and Facilities/Hardware management [Ref. 34].

2. Engineering Change Proposal - Used in submitting proposed changes to baselined software configuration. Either DD Form 1692 or 1693 is used [Ref. 34].

3. Request for Deviation - This is used when it is necessary to depart temporarily from documented requirements. DD Form 1694 is used in this case [Ref. 34].

4. Request for Waiver - If an item fails to conform to its required configuration and this is due to a development error, a Request for Waiver is submitted. This is also submitted on a DD Form 1694 [Ref. 34].

5. Engineering Change Orders - Once approval has been granted, Engineering Change Proposals, Deviations, and Waivers are implemented through an Engineering Change Order [Ref. 34].

6. Specification Change Notice/Notice of Revision - Both these documents are used when an Engineering Change Proposal or Waiver affects baselined documents or drawings. DD Form 1696 is used for the SCN and DD Form 1695 is used for the NOR [Ref. 34].

7. Software Trouble/Incident Report - This is used for reporting all deviations from test procedures, equipment malfunctions and software anamolies [Ref. 36]. These become a part of the software Test Report and a permanent part of system documentation.

h. Program Library Control

Program Library Control is designed to maintain and control a systems computer programs and all changes to these programs. Software Quality Assurance personnel are responsible for approving all changes to library programs. This system not only provides baseline, error-free, patch free programs for test and evaluation but it also insures that all approved changes are incorporated into the programs [Ref. 37]. TRW uses a system similar to this.

i. Benefits

In [Ref. 33] it states that NOSC has realized the following benefits through the establishment of its Software Quality Assurance program:

1. It establishes a controllable structure in the software development process.
2. It assures certain elements of quality in every phase of the development.
3. By reducing rework substantially, it provides for the satisfaction of requirements.
4. The substantial reduction in the amount of rework has lead to a significant savings in life-cycle costs.

3. General Electric Company

a. Introduction

The software quality assurance activities at two separate divisions of the General Electric Company will be discussed. One discussion will cover the Electronic Systems Division located in Syracuse, New York. The other will cover the Space Division located in King of Prussia, Pennsylvania. Neither discussion will be extensive as both divisions use

quality assurance techniques which are similar to the two organizations previously described.

b. Electronic Systems Division

(1). Quality Assurance Objectives. John McKissick Jr. and Robert A. Price point out in [Ref. 38] the objective of the Computer Software Quality Assurance (CSQA) Program at the G. E. Electronic Systems Division is to ensure that software delivered under a contract meets the requirements of the contract. As can be seen this is similar to the organizations previously discussed.

(2). Quality Assurance Planning. The manager of Computer Software Reliability and Quality Assurance and dedicated technical specialists are responsible for planning and implementing the QA Program. In addition to providing staff support to project managers the manager of Computer Software Reliability and Quality Assurance reports directly to the manager of Reliability and Quality Assurance. It should be noted that the Computer Software Reliability and Quality Assurance group is organizationally independent of Software Engineering.

After reviewing both the Computer Program Management Plan and the Software Standards and Procedures Manual, Quality Assurance personnel develop the Computer Software Quality Assurance Plan [Ref. 38]. The plan defines all activities which control and assure computer software quality. The plan is developed using Mil-S-52779A, and it identifies the organizational component responsible for each activity.

(3). Software Standards. Similar to TRW the G. E. Electronic Systems Division also develops a Software Standards and Procedures Manual. This document, which is primarily used by the programming teams, establishes rules, guidelines, and limitations which are to be observed in

generating software designs and code which will have the properties of: consistency, readability, and quality [Ref. 38]. The structure and content of the Software Development Notebook (SDN) is also defined.

The SDN is similar to the Unit Development Folder used by TRW. It is a simple loose-leaf notebook which is established during the Preliminary Design phase for each Computer Program Component (module). It provides a common collection point for all information dealing with a CPC and it is maintained and updated throughout the remaining phases of the software development [Ref. 38].

The SDN is broken down into the following sections: Requirements, Detail Design, Functional Capabilities, Code, Test Case Descriptions, Test Case Results, Software Problem Reports, and Miscellaneous Information. With the exception of Software Problem Reports and Miscellaneous Information, each section contains a cover sheet showing schedule dates, actual completion dates, and review and approval signatures [Ref. 38].

The SDN, like TRW's UDF, serves as the principle working document of the programmer. In addition it provides management, the customer, and QA personnel visibility into the design, status, and quality of the software under development [Ref. 38].

SDNs are audited monthly. This audit may be on either an announced or unannounced basis.

(4). Quality Assurance Reviews and Audits. Audits and reviews similar to those conducted by both TRW and NOSC are conducted at the G. E. Electronic Systems Division. Internal reviews are conducted by software and systems engineers who have not contributed to the design under review. The review chairman is responsible for implementing the review plan which was approved by Quality Assurance and the Project Manager.

After the internal reviews are conducted, joint customer/contractor reviews are held. These reviews provide a technical forum for better mutual understanding of the performance requirements allocated to the computer software, and of the design approach selected to meet these requirements [Ref. 38].

(5). Testing. At the G. E. Electronic Systems Division the Test Plan is developed during the preliminary design phase of the development process. Test procedures are developed during the detailed design phase. Actual testing occurs in the final four phases of the development process: Code, Debug and Unit Test, Development Testing, Integration Testing, and Acceptance Testing [Ref. 38].

Unit testing is conducted on individual routines to reveal coding errors, computational errors, improper input handling, inappropriate error messages, and incorrect formatting and content of output. Development testing takes the prviously tested routines and combines them to form Computer Program Components (CPC). The functional capabilities of the CPC is are then verified. Integration testing, which is performed by an independent test team, combines CPCs and verifies the correct sequencing of components, compatible component interfaces, and proper data routing. Acceptance testing, which is also done by an independent team, verifies system level functional requirements. These include overall timing and the ability to handle the total input load. As at TRW and NOSC QA personnel witness all testing [Ref. 38].

(6). Problem Reporting and Review. Like both TRW and NOSC, G. E. Electronic Systems Division has a Configuration Control Board which reviews and approves all changes to the software. As in both the previously described organizations, Quality Assurance personnel are members of this board. Quality Assurance personnel are also responsible

for verifying that all approved changes have been incorporated. This is similar to both NOSC and TRW.

A Software Problem Report (SPR) defines and documents a problem and the test conditions under which it occured. Quality Assurance receives a copy of all SPRs. In addition they maintain a listing of all outstanding SPRs and the party responsible for corrective action. Once a problem has been corrected Quality Assurance annotates its copy of the SPR with the way in which the problem was corrected [Ref. 38].

All coding changes are authorized using a Software Change Order (SCO). Changes to hardware and software specifications are documented on a Specification Change Notice (SCN) [Ref. 38].

(7). Benefits. An effective Quality Assurance Program allows the G. E. Electronic Systems Division to deliver computer software which meets all contractual requirements. In addition it provides management visibility into the software development process.

c. Space Division

(1). Quality Assurance Objectives. The Quality Assurance Program which was reviewed has been in effect at the General Electric Company's Space Division since 1978. The primary objective of the program is to ensure that delivered software meets all contractual requirements. A secondary objective, which is designed to help secure contracts, is to define and implement specific measures designed to ensure delivered software incorporates the features necessary to achieve testability, maintainability, reliability, etc. [Ref. 29].

(2). Quality Assurance Planning. As is the case in the three previously described organizations the primary job of the Quality Assurance group is to prepare the

QA Plan. Once the QA Plan has been developed Quality Assurance turns its attention to the implementation and management of the plan.

(3). Software Standards. Each programmer working on a given project receives a copy of the Programming Standards Document (PSD), which outlines the standards to be used to produce a high-quality software product. It is not enough however that the programmer be given the PSD and left to go along his merry way. Training sessions, conducted by the Software Development group, are held to explain the content of the PSD. It is the job of the Quality Assurance group to verify that each programmer participates in the training sessions [Ref. 29].

(4). Quality Assurance Reviews and Audits. The G. E. Space Division conducts reviews and audits similar to those of the previous three organizations. They do, however, give special attention to the developmentof software interfaces. The development of the software interfaces is done by the Software Development Group. However the System Engineering group is responsible for developing the Interface Control Documents (ICD). The purpose of this process is to provide a timely and complete definition of interface details and to provide a continuous in-depth review process [Ref. 29].

(5). Testing. The Quality Assurance role in the actual testing is minor. The bulk of the Quality Assurance groups work is done prior to actual testing.

Quality Assurance first identifies exactly what is to be tested and at the same time develops a detailed definition of the test environment. They then explicitly define both the test and the evaluation process, especially success/failure criteria.

During the actual testing, Quality Assurance acts as a monitor to ensure that the previously defined procedures are being followed. They also ensure that any changes are documented correctly [Ref. 29].

(6). Problem Reporting and Review. Any problems uncovered during the testing process are documented using a Discrepancy Report (DR). Once testing is complete a post-test meeting is held and the Discrepancy Reports are assigned to individuals for resolution. The Software Quality Assurance group monitors all outstanding DRs to ensure that all problems are corrected.

The Discrepancy Reports serve as a measure of product quality. The software QA group analyzes the data provided by the DR and prepares a statistical report. In [Ref. 29] Stephen L. Stamm, Manager of Productivity Programs at the G. E. Space Division, points out that such things as the number of DRs, the frequency distribution of DRs by type, the mean time of closure (correction), and the DR rate as a function of product life can be used by management to identify weak spots in the software implementation process.

(7). Program Library. A system similar to that at NOSC is used at the G. E. Space Division.

(8). Quality Assurance Tools. Automated software code analysis tools are used as part of the test program to uncover code in the final product which has never been executed. If this is the case it is determined if there is a hole in the test program, a possible flaw in the product design or just some superfluous code in the finished product [Ref. 29].

(9). Lessons Learned. The following elements [Ref. 29] have been found by the G. E. Space Division to be necessary for a successful Software Quality Assurance Program:

1. The Software Quality Assurance Plan must have high project visibility, it must define the SQA program at

a level of detail sufficient to allow implementation, and it must have the project and company managements whole-hearted support.

2. The application of special software engineering techniques by the programming staff specifically targeted at increasing product quality.

3. The project organization must distribute the Software Quality Assurance Program responsibility, placing SQA tasks where the capability really exists.

4. The ability to measure the effectiveness of the QA program and, if possible, the quality of the end product.

5. Software Quality Assurance personnelmust be an integral part of the project team.

E. CONCLUSION

Four separate Software Quality Assurance organizations have been discussed in this chapter. However they are similar in several ways.

First the Software Quality Assurance groups at each of these organizations become involved with the project early in its life-cycle. Each of the Quality Assurance groups is involved in every phase of the development process. Each of these organizations recognizes the fact that an effective Quality Assurance organization allows them to deliver a quality software product which meets all contractual requirements. They are also in agreement on the fact Software Quality Assurance saves money in the development process by identifying and correcting errors early in the process. They also agree that an effective QA program provides both management (Corporate and Project) and the customer with visibility into the development process. The view of software as a product is also a similarity among these organizations. Finally all these groups agree that Software Quality Assurance groups do not create quality in a project but that it is in fact part of the job of every project member.

IV. THE QUALITY APPROACH AT FMSO

A. STRUCTURE

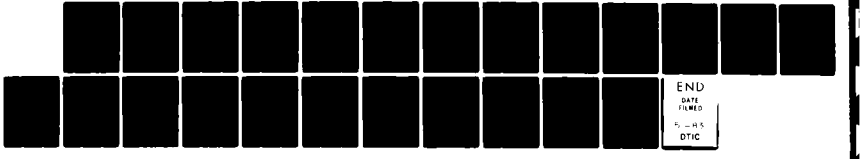
Within the Naval Material Command, the Fleet Material Support Office (FMSO) is a field activity sponsored by the Naval Supply Systems Command. FMSO performs two major functions in fleet logistical support. The one discussed here is that of principal Navy Central Design Agency (CDA) for automated supply, financial, maintenance, and logistical systems, a process which consumes the majority of FMSO resources.

A general description of the organizational elements directly related to the central systems development process is depicted in figure 4.1 Within the organization the Comptroller Department (Code 91) and Management Department (Code 92) are the two departments that can be considered as staff. The other six CDA departments are production oriented or support the production effort and are considered as line organizations which are directly responsible for the development and maintenance of standard Automated Data Processing (ADP) systems [Ref. 39].

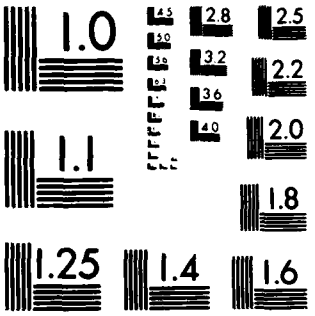
A system is considered to be an organized set of ADP hardware, environmental/application software, and documented procedures designed to automate the basic management and operating processes for a customer site or group of customer sites with common mission responsibilities. "Documented procedures" as used above refers to the applicable ADP related and non-ADP related procedures established to support the hardware and software aspects of the system [Ref. 39].

AD-A127 676 A SURVEY OF SOFTWARE QUALITY ASSURANCE METHODS AND AN EVALUATION OF SOFTW..(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M FUQUA ET AL. DEC 82 2/2

UNCLASSIFIED F/G 14/4 NL



END
DATE FILMED
5-83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

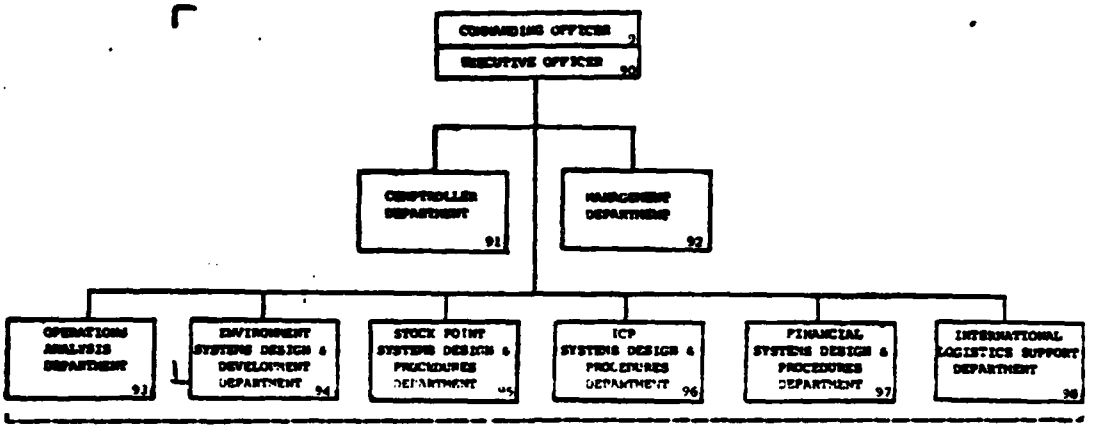


Figure 4.1 Organizational Structure.

The primary role of the Management Department is to coordinate with and support the efforts of the CDA Production Departments. In this capacity the branch that is of most relevance to this paper is that of the Quality Control Branch. The CDA Development Process Model, figure 4.2, reflects all of the basic steps appropriate to ensuring that each CDA tasking received by FMSO is effectively managed and results in a high quality product being released for use by the customer. The model covers all projects, large and small, new developments or maintenance. However, it is anticipated that some of the steps in the model may not be applicable to all projects. Therefore, an explicit decision by the appropriate level of management is required in order to exclude process steps determined not applicable on a project. As the model is followed through, note specifically the areas that deal with symbols equating to 92 QC and 92 QC Optional [Ref. 40].

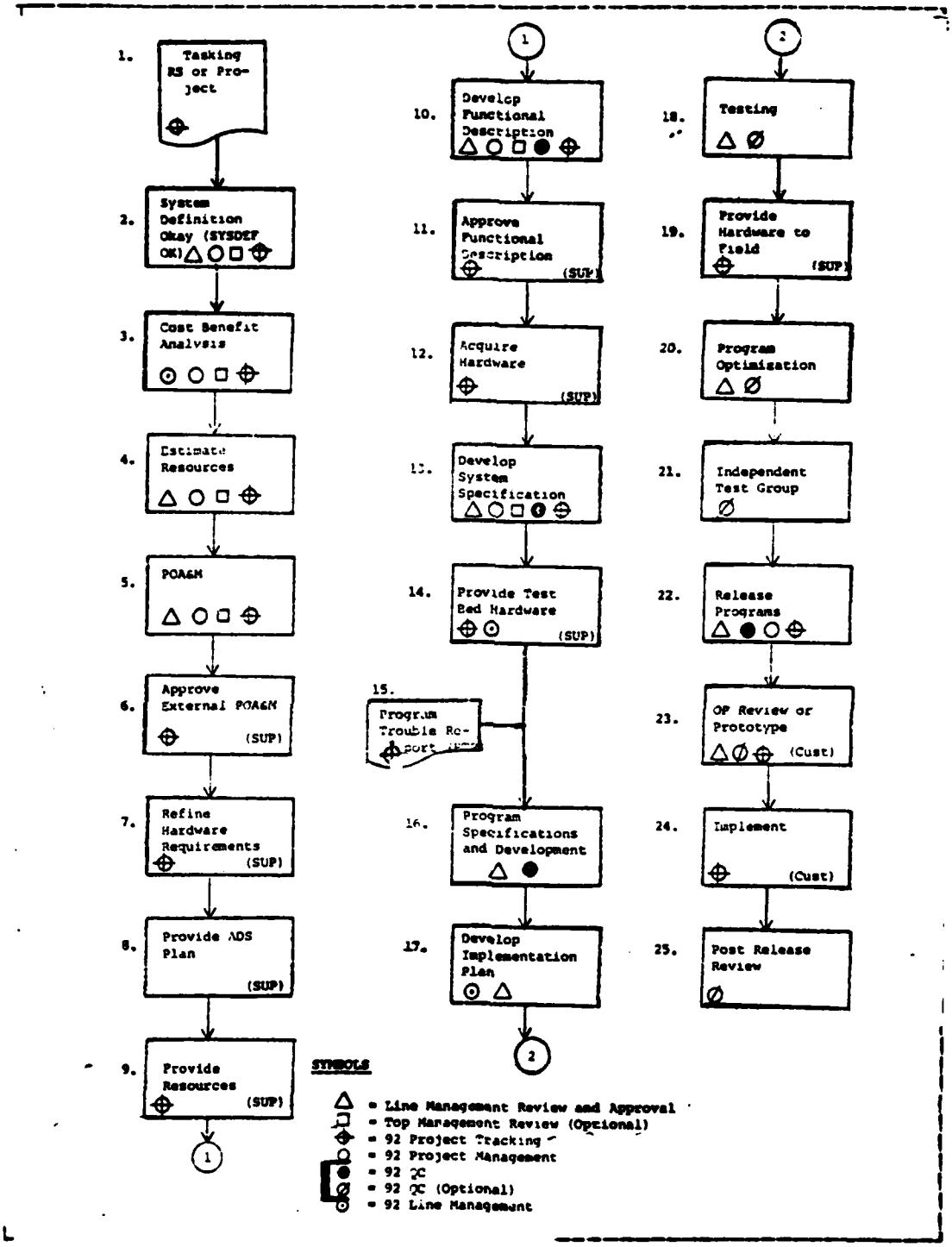


Figure 4.2 Project Flow Model.

B. THE QUALITY PROCESS

As a new requirement is received by FMSO a mechanism is activated to ensure that the output produced will meet the user's expectations. This mechanism at FMSO is called the quality process, that is, an attitude that extends from the individual programmer all the way through the systems development cycle up to top management. By design, it is a multi-layered approach to achieving quality that begins with the System Development Quality Process (SDQP). Falling under the SDQP are all the separate requirements that will eventually produce a working ADS. At each stage of development, quality standards are imposed upon all personnel at all levels within the chain of command beginning with the feasibility study and requirement definition, proceeding through the functional design, computer design, program development, testing, operations and maintenance, and ending with the Program Trouble Reports (PTR). Within each of these particular evolutions labeled above are subsections that must be completed before the process evolves further.

Layered on top of the SDQP are the Quality Control Mechanisms. These Quality Control Mechanisms provide the ability to ensure that a quality and error free product is in fact produced. The methods utilized to accomplish this are good project management during the requirement definition and functional design stage, sound data base management during functional design and computer design, an effective verification process during computer design and program development, proper validation procedures during program development and testing, and a satisfactory prototype/op review during testing and operations/ maintenance of a system.

On top of both the SDQP and Quality Control Mechanisms we have the Quality Production Concepts. These include structured processing, standard data element usage, uniform standards, methods and procedures, improved programming techniques and training. When all of these various layers are utilized and implemented as a whole we have the philosophy of FMSO towards producing a quality product.

C. QUALITY ASSURANCE VS. QUALITY CONTROL

Quality Assurance, as defined by FMSO [Ref. 40], is a line management responsibility. As such, Line Supervisors are accountable for enforcing the application of standard procedures that have been developed for the primary purpose of insuring accuracy, thoroughness of method, simplicity in design, adequacy of testing and clarity of documentation of ADS development. To aid all levels of personnel within the various line departments in the accomplishment of their own specific requirements for quality, guidelines have been developed that include NAVSUP PUBs 506, 507, 508, CDA DEVELOPMENT HANDBOOK, CDA MANAGEMENT HANDBOOK, FMSO Internal Instructions, and various other documented and undocumented departmental procedures. Everything written and documented must be in compliance with these standards. It is the individual person's responsibility to ensure that they are in compliance with these standards.

Quality Control, as defined by FMSO [Ref. 40], is the responsibility of the Management Department and specifically the Quality Control Branch. Quality control procedures are those actions that are taken as an ADS is being developed to insure that all the required QUALITY ASSURANCE procedures , those actions performed by the line department personnel, will be complied with to produce a reliable and error free ADS product. Quality Control then is primarily a review

function to be performed by the Quality Control Branch. In this area they are responsible for ensuring that designated/high interest ADS projects conform with standards of completeness, accuracy, clarity, and all other applicable quality standards that applied to the line quality assurance program have been achieved.

D. SPECIFIC QUALITY CONTROL RESPONSIBILITIES

As quality control is a review function and given the limited number of personnel assigned to the branch, not every output that is provided by FMSO will be reviewed by the Quality Control Branch. The Quality Control Branch will, however, be directly involved with those projects designated high priority or of special interest to the Command. All other projects will be reviewed on a priority basis as the manhours that can be devoted to the futher enhancement of the quality effort become available. Their area of responsibility is enormous and their tasks numerous. Therefore, only areas of major responsibilities are listed below [Ref. 40],

1. Review of Functional Descriptions for compliance with standards.
2. Participate in a system design review to ensure that the design has considered all of the proposed system requirements.
3. Review of System Specifications for compliance with standards.
4. Review of Program Specifications.
5. Review of the Maintenance Manual, Operations Manual, and all other applicable manuals.
6. Prepare an analysis of PTRs received as to cause or symptom and recommend possible corrective action.
7. The Quality Control Branch will selectively review test plans and tests for compliance with Quality Assurance guidelines. In the performance of this task, they may desk check all applicable data or they may elect to attend the review conference held between the programmer and analyst as they discuss the results of the test.
8. Review the Implementation Plan.

9. Quality Control is responsible for post implementation visits to selected sites of designated projects to determine whether the product released is working satisfactorily, meets the needs of the user, and is being utilized correctly.

10. Perform a Quality Assurance Review of all designated ADS programs.

E. TESTING TO ENSURE QUALITY

The testing process involves many different personnel and includes many different responsibility assignments. For example, the individual programmer assigned to the project is responsible for reasonable testing for all error conditions that could occur in the program and for providing support for the systems test required for the application. The Lead Programmer is responsible for developing the test plan for system testing and/or string testing.

The Systems Analyst is responsible for assisting the Lead Programmer in planning and coordinating the string testing/system testing to determine that all the programs in the application produce the required output when run in total. The Systems Analyst has a primary responsibility of approving and/or selecting the test data used for systems testing.

The Systems Designer is to participate in reviews of test output to insure that testing of the ADP program has been adaquete. After the processes above are completed, the Quality Control Branch will selectively review test plans and test results for compliance with all Quality Assurance Guidelines.

1. Types Of Testing

During the testing cycle there are primarily four different methods utilized to check the programs.

1. Unit Test - A single program that is checked by the programmer responsible for also writting the code. A

Unit Test Review is scheduled between the analyst and programmer with the test results being reviewed to ascertain if futher testing is warranted.

2. String Test - Each program release which requires the execution of other programs in actual production will be string tested. Programmers are responsible for writing the test plan to ensure that all major paths and functions that will utilize the new program are checked.
3. System Test - Each new ADS or major change involving more than one application/operation or package in an existing ADS will be subjected to a system test that will evaluate the specific system as a whole. The overall responsibility for the test will be given to the Lead Programmer. The Quality Control Branch will evaluate the results to ensure that the system meets design objectives.
4. Integrated Systems Test - This test will be designed to test all program interfaces, database interfaces, and all internal and external applications for correctness of data flow. The Lead Programmer and Lead CDA Department are responsible for preparing a formal test plan and conducting the test. The Quality Control Branch may or may not be assigned as an overall monitor for this procedure but will be required to review the results.

F. SYSTEM RELEASE PROCEDURES

Upon completion of the required evaluations and for specified projects, the Line Departments involved will forward to the Quality Control Branch all applicable documentation for review. After this review has been completed the complete package is sent back to the CDA Department Line

Manager. When the Manager is satisfied that the program meets the requirements and that all quality assurance standards have been met, the Line Manager will, by his signature release the program for use. This will then terminate the Systems Development Process.

G. EVALUATING THE QUALITY PROGRAM

It is important to realize that at FMSO, Quality Control is not intimately concerned with the daily operations of the line departments, but is concerned with assessing the results of the line departments. With this perspective in mind, the design and execution of the Quality Assurance Plan is considered to be an integral part of the production process itself. Quality assurance for software consists of the formal application of standards and execution of required tests, and then the assessment of results. Enforcement of the standards and the necessary adjustments of the process is one of the responsibilities of the Quality Control Branch.

Although the Quality Control Branch is deeply embedded within the Management Department, they have performed in an extremely professional manner. When given an adequate amount of time and an opportunity to perform in their primary role as reviewers, they have always met the challenge. But this opportunity to excel does not always occur as it should, and FMSO is no different than any other software producing organization. As the project completion date draws near, usually the first item to be called excessive to the program is quality control. At FMSO, the project manager is responsible for ensuring that enough time is alloted during the SDP so that Quality Control has a chance to evaluate the program. This estimate of the amount of time it will take to complete a project is a most difficult

one, but dates have to be set and, on occasion, met. The authors are unable to cite specific examples of Quality Control being cut short, but when the amount of work that is to be accomplished and the number of personnel assigned to accomplish it are compared, the assumption can be made that it has occurred , either as a result of internally or externally generated pressure to comply with a due date.

When the value of a product is very high, such as on command designated /special interest projects, each item produced is individually inspected and gone over in fine detail at all levels, from the programmer through all the reviews, and finally top management. However, under less important conditions, the selection and review process of products is not as critical, and for very minor projects it need not be. This does not mean that the product is any less important to the user but merely implies that all projects other than the exceptionally simple ones should also receive the same degree of scrutiny that special projects do.

The reality of the present situation dictates that the above process is not feasible at this time. With such a small staff in the Quality Control Branch and the multitude of tasks assigned, it is physically impossible to meet all of their requirements, and priorities must be established. With an ongoing review of the number of Computer Specialists that can be justified within the Quality Control Branch, it is essential that the justification be met and billet descriptions constructed to place more people, not less,in this most important branch if FMSO is to continue with its present emphasis upon producing a quality product.

The authors feel that the Quality Control/Quality Assurance program at FMSO is highly competitive with similiar organizations. Their thorough and most agressive instructions and standards are excellent, and if the present

level of enforcement is continued will greatly enhance the product serving the Fleet. Quality Control/ Quality Assurance does happen at FMSO, primarily due to the structured process and to the professionals that work and manage the organization. The degree to which it happens is an evolving entity.

At the present time the only effective documented method to measure the application of quality practices within FMSO is the analysis and evaluation of the Program Trouble Reports (PTR). PTRs may be submitted during any step of the systems development process or after the program has been sent to the field user. As a PTR is received at FMSO it is routed to the Project Control Branch where is is logged in and then sent to the department that issued the program with which the PTR is concerned. The department involved then decides if the PTR is of a critical or non-critical nature and then proceeds to work on it. PTRs are classified in two ways, critical, which has a significant impact upon daily routine, and non-critical, which has less of an impact and can temporarily be delayed. Critical PTRs will be corrected as soon as possible and normally within three working days from receipt of sufficient information to allow the CDA to act. Non-critical PTRs will be corrected as soon as possible after receipt of sufficient data that will allow the CDA to act.

The Quality Control Branch performs a quarterly analysis of all PTRs received with special emphasis on the most common type of problem, which steps of the systems development process create the most errors, identification of trends, and if possible, recommendations for corrective actions.

A survey of the PTR Analysis Report for the Second Quarter CY 82 reveals the following: [Ref. 41],

1. A total of 385 PTRs, of which 275 were completed, 92 were cancelled, and 18 reclassified. Those cancelled were due to either being invalid or already in existence. A reclassified PTR was one that, when submitted, it was felt by the user that the program was not performing as desired or requested but upon researching the problem it was found that all requirements had been met. To meet the new requirement of the user a new project would have to be designated.
2. Of the completed PTRs, 38 were critical and 237 non-critical.
3. The PMSO average number of manhours to fix a PTR was 16 for a critical PTR and 21 for a non-critical PTR. One possible reason for the difference in times is the experience level of personnel assigned to repair a program. A more experienced programmer is generally assigned to a critical PTR.
4. The average number of days for PMSO to complete a critical PTR is 15.5 and 151.6 to complete a non-critical PTR. This figure may be skewed toward the high side very easily because of the small number of critical PTRs but does bear close monitoring.
5. Of the completed PTRs, 57 percent were caused by coding/design errors. 21 percent were classified as other and not designated to any category. However, 42 percent of the critical PTRs were caused by program or coding errors. Steps have been taken to better divide the cause category in an attempt to better evaluate the errors that were classified in the other category.
6. The comparative completion rates for critical PTRs has remained relatively stable compared to the past year. However, the non-critical completion rate has

increased substantially and the trend is for an even greater completion rate.

7. The Received-Resolved-Outstanding results show that the number of PTRs received is leveling off, the number of PTRs resolved is increasing drastically, and as a result of both of these the number of outstanding PTRs is now in a steady decline.
8. The number of programs released increased about 12 percent over the previous quarter and yet the number of PTRs received decreased by about 7 percent. Although the number of PTRs over the last year has oscillated, the trend shows there to be no significant increase and thus a net decrease in ratio of programs to PTRs should be anticipated.

Considering that only nine personnel are assigned to the Quality Control Branch and that FMSO now has 10,000 plus programs in existence, the Quality Control/Quality Assurance plan appears to be headed in the right direction as the PTR report clearly shows. With continued emphasis on the Quality Control effort an even more effective program will be displayed in the future. To achieve a 100 percent error free product is the ideal but a more realistic goal must be set and an effective method of measuring and selecting the programs for a more detailed investigation will aid greatly in accomplishing the goal FMSO sets for itself. The PTR report shows an improvement over the preceeding year but it also quite effectively shows other areas that may need additional emphasis. In the following chapter we will list some of the areas that we feel should be considered in the future corporate growth pattern of FMSO.

V. RECOMMENDATIONS

This paper has been a comprehensive investigation of software quality assurance from a theoretical viewpoint and from specific investigation into the quality assurance/ quality control departments of various organizations. Based on the authors' research, the following recommendations are offered in hopes of enhancing the Fleet Material Support Office's (FMSO) quality control effort.

1. A corporate base line concerning the quality process needs to be established. Before a specific direction can be maintained, a mechanism must exist that would enable FMSO to measure its ability to meet the desired objectives. It is felt that, at a minimum, the Quality Control Branch could examine past programs and select at least two that were very similar for an analysis as to the effectiveness of the quality control program. The criteria of these two programs might be (1) that they were produced by the same department within a short time period of each other, (2) that they were intended to be utilized in the same fashion, and (3) that one of them had been reviewed by Quality Control throughout the entire process and the other had not received this same critical review.
2. Top management must continue to re-emphasize the importance of quality control within the organization and display a positive philosophy of commitment to the quality process. As has been discussed, without top management support, quality assurance/quality control programs produce a less than desirable output.

3. The number of personnel assigned to the quality control branch is definitely inadequate. In an organization as large as FMSO, which produces 800 plus programs per quarter, a quality control organization of only eight people plus a supervisor can not reasonably be expected to meet their obligations and responsibilities on all occasions. The Quality Control Branch needs an infusion of personnel. Attention should be paid not only to the quantity of personnel but also to quality. As stated previously, the core of these personnel need to be as knowledgeable as senior systems analysts and also command the respect of the individuals whose systems are being evaluated.

4. A method of augmenting the staff of the Quality Control Branch from an external source may be to utilize it as an indoctrination facility for new hires. A core of highly qualified personnel could be maintained that were the permanent part of the Quality Control Branch. New hires could be given this temporary position and tasked with such projects as reviewing the project specifications, all of the manuals, and all other applicable material. This would also afford the new personnel an opportunity to receive formal and correct training before being assigned to the particular line department. Additionally, it would enable the new hires to gain a better overall understanding of what the organization does and the amount of interfacing that must conducted before a project is completed. Of course there would have to be some discretionary measures imposed when selecting personnel and a time limit must be adhered to.

5. Consideration should be given to moving the Quality Control Branch out of its present management structure. The authors feel that, because of the massive size of the organization and enormous amount of material that must be reviewed, the Quality Control Branch should be moved so as to have at least parity with line management. This position may be designated as Quality Control Department Head, Code 99. Billet descriptions would have to be re-written and the management problems overcome, but the increased communications, be they voluntary or by direction, between quality control and line management would have an impact upon future products. Additionally, this would afford the quality control personnel an easier avenue to become more directly involved with their responsibility of ensuring that standards are met and better enable them to enforce these standards.
6. Line managers must be educated as to the importance of the quality control function. In this vein, when openings arise in the Quality Control Branch, qualified line personnel should be encouraged to apply for the positions. The benefit gained for the organization would more than offset the loss to one individual department. To support the attitudinal change that must take place, an ongoing training program, sponsored by top management, will have to be implemented on a company wide basis and the positive benefits gained from the addition of these qualified personnel to the Quality Control Branch must be discussed and displayed.
7. Quality control checklists must be re-instituted. There should be a general checklist applicable to all programming functions as well as specific checklists

regarding each individual program. It should be emphasized that these checklists are to be used as reminders or as a constructive tool to reinforce standards, rather than a method of laying blame.

8. It is important for continuity that there be consistent assignment of quality control personnel to projects. One person (or team of persons) should be assigned to a project, except the smallest ones, and should monitor this project all the way from inception to post implementation review.
9. A better method, possibly a decision support system, must be devised to determine which projects will undergo quality control procedures. It is recognized that the number of projects may preclude all projects being scrutinized by quality control. However, there needs to be a better system to select programs for review and evaluation than the present method of committee or command discretion. Currently, the primary method for evaluating project length and complexity is experience. This subjective viewpoint is the basis for the depth of involvement achieved by quality control. A logical and comprehensive decision process will serve to alleviate crisis management and ensure that the most important projects are chosen.
10. End users must become even more involved in the design and development of software throughout the lifecycle. Users should work closely with quality control to ensure a timely and correct review process. There must also be better communications established between users and line programming functions. This communication may be facilitated by quality control and may help lower project trouble report (PTR) reclassifications.

11. Clear and concise documentation is an ongoing problem at any software production facility. The quality control division must review documentation, not coding, and assist where possible to improve any deficiencies.
12. Currently, the program trouble reports (PTR), are the primary documented measure of the effectiveness of the quality program. While this may be a valid measurement, there needs to be a search for additional measures. Since quality control does not impact 100 percent of the programs leaving FMSO, a method of evaluating the effect of quality control on projects which do not have quality control involvement must be formulated.
13. The principle of top-down design and top-down testing should be reevaluated. As an alternative, top-down design and bottom up testing should be considered. It is assumed that top management will feel uneasy as the present process is changed. However, it has been shown in many studies that errors made in the design phase of a project are the most expensive to repair because one has to return back to the beginning and literally begin the entire process over. Requirements must be reevaluated, specifications redone, coding rewritten, and finally, the program must be tested again. Since the majority of design errors are not found until the testing phase, it is easy to see that the amount of extra time spent in the design phase will, over time, offset any anxiety that top management would have because of the seemingly lack of progress on the project. It is suggested that bottom-up testing will force the design effort to improve.

14. Phased development should be adhered to. Coding must not begin until after the design is complete. By doing so the lead programmer can assign the better programmers to the difficult modules and an inexperienced programmer to the easier modules/programs. This line of thought should filter throughout the project and allow for better personnel utilization, but should also afford the persons writing the test plan to do so in a more reliable manner. In allowing the writing of code before the final design is completed, we achieve the short term goal of being able to see a working product that can be tracked on a chart. However, we cannot see the long range goal of whether the module will produce the required output or the required number of interfaces.
15. The system to trace the phase of development for each project should be reemphasized and quality control must continue to be kept abreast of all the current production efforts.

LIST OF REFERENCES

1. Ross, Douglas T., "Quality Starts with Requirements Definition", Constructing Quality Software, Edited by P. G. Hibbard and S. A. Schuman, North-Holland, 1978, p. 397-401.

2. Johnson, K., "The Software Crunch," Computer Management, p. 25, 27, 30, October 1981.

3. Cavano, Joseph P., "A Framework for the Measurement of Software Quality", Proceedings of the Software Quality Assurance Workshop, p. 134, 1978.

4. Perry, William E., Effective Methods of EDP Quality Assurance, Q.E.D. Information Sciences, Inc., Wellesley, Massachusetts, 1981.

5. Schultz, Brad, "Study Covers Six Steps to Quality Assurance", Computerworld, February 15, 1982.

6. Boeing Aerospace Company, Software Quality Assurance-One of the Software Acquisition Engineering Guidebook Series, January 1979.

7. Buckley, F. J., "A Standard For Software Quality Assurance Plans - A Status Report," Proceedings of the International Conference on Computing in Civil Engineering, First, New York, p. 1178-1187, 1981.

8. Robertson, A. G., Quality Control and Reliability, Thomas Nelson and Sons Ltd, 1971, p. 7.

9. Hansen, S.D. and McHarg, J.D., "Reliability and Quality Control of Production Engineering Computer Programs," J. Aircraft, Vol. 11, J. Aircraft, Vol. 11, No. 4, p. 232-236, April 1974.

10. "Software Development Costs", Digital Design, p. 24, June 1980.

11. Goldsberry, V. W. and Tibodeau, A. K., "Establishing a Software Quality Assurance Function in a Traditionally Hardware Organization," MIDCON '79 Technical Papers, Vol. 3, Paper 25.1.

12. Prudhomme, R. R., "Software Verification and Validation and SQA," ASQC Technical Conference Transactions-Atlanta, p. 397-405 May 20-22, 1980.

13. Roberts, T. J., "Maintaining Quality After the Software is Released", ASQC Technical Conference Transactions, p. 157-167, May 1977.

14. Howley, P. P., "Software Quality Assurance for Reliable Software", Proceedings of IEEE Annual Reliability and Maintainability Symposium, p 73-78, 1978.

15. Howley, P. and Fink, A. J., "Software Quality Assurance Standard," Boeing Aerospace Company document D180-17663-1, January 1976.

16. National Bureau of Standards Special Publication 500-11, Computer Software Management: A primer for Project Management and Quality Control, by D.W. Fife, p. 13-15, July 1977.

17. General Electric Co. and Rome Air Development Center Report F30602-78-C-0216, Software Quality Metrics Enhancements, Vol. I, by J. A. McCall and M. T. Matsumoto, p. 16, April 1980.

18. Forrester, M. H., "Quality Control in Software," Quality Assurance, Vol. 5, No. 4, p. 99-104, December 1980.

19. Dunn, Robert H. and Ullman, Richard S., "A Workable Software Quality/Reliability Plan," Proceedings of the Annual Reliability and Maintainability Symposium 1978.

20. Goodenough, John B. and McGowan, Clement L., "Software Quality Assurance: Testing and Validation," Proceedings of the IEEE, Vol. 68, No. 9, September 1980.

21. Srinvasan, C. A. and Dascher, P. E., "Quality Assurance Program - A Method to Improve", Hospital Financial Management, Vol. 35, No. 6, p. 25, June 1981.

22. Department of the Army Military Specification MIL-S-52779A, Software Quality Assurance Program Requirements, 1 August 1979.

23. Buckley, F., "A Standard for Software Quality Assurance Plans," Computer, p. 43-50, August 1979.

24. Fujii, M. S., "A Comparison of Software Assurance Methods," Proceedings of the Software Quality Assurance Workshop, p. 29, November 1978.

25. Mendis, K. S., "A Software Quality Assurance Program for the 80's," ASQC Technical Conference Transactions, p. 380-389, May 20-22, 1980.

26. Warren, J. H. and Turpin M. P., "Design and Development of Reliable Software," Quality Assurance, vol. 5, no. 4, p. 117-121, December 1979.

27. Shirey, R. W., "Quality Assurance Tools," Computerworld, Vol. 14, No. 20, In Depth p. 31-49, May 19, 1980.

28. Ridge, W. J. and Johnson, L. E., Effective Management of Computer Software, 1st ed., p. 142, Dow Jones-Irwin, Inc. 1973.

29. Stamm, S. L., "Assuring Quality, Quality Assurance", Datamation, v. 27, no. 3, p. 195-200, March 1981.

30. Fischer, K. F., "A Program for Software Quality Assurance," ASQC Technical Conference Transactions, p. 333-340, 1978

31. TRW Defense Systems Group, Quality Assurance Program Plan for the Sentry Sensor Engagement Controller, by J. F. Rogers, p. 3-1 to 3-12, 15 July 1982.

32. TRW Defense Systems Group, Configuration Management Plan fo Sentry Engagement Controller, by J. F. Rogers, p. 3-1 to 3-16, 15 July 1982.

33. Naval Ocean Systems Center Technical Note 410, Software Quality Control Practices Manual, p. 1-1 to 4-13, 1 May 1978.

34. Naval Ocean Systems Center Technical Note 413, Procedure for Software Configuration Management (R&D), by A. T. Charlesworth, p. 1-17, 1 May 1978.

35. Naval Ocean Systems Center Technical Note 419, Procedure for Software Configuration Reviews and Audits, by J. F. Parlier, p. 3-14 to 3-30, 1 May 1978.

36. Naval Ocean Systems Center Technical Note 415, Procedure for Software Test and Evaluation, by G. D. Tice, p. 4-1 to 4-3, 19 September 1978.

37. Naval Ocean Systems Center Technical Note 416, Procedure for Computer Program Library Control, by K. M. Koch, p. 2-1, 1 May 1978.

38. McKissick, J. Jr., Price, R. A., "Quality Control of Computer Software," ASQC Technical Conference Transactions, p. 391-398, May 1977

39. Fleet Material Support Office, CDA Management Handbook, Chap. 2, 1 Jan. 1981.

40. Fleet Material Support Office, CDA Development Handbook FMSOINTINST 5230.20a, Change 3, Ch. 2, 1 Feb. 1982.

41. Fleet Material Support Office, PTR Analysis Report: Second Quarter, CY 82, 4 Aug. 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 54 Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
4.	Professor Norman R. Lyons, Code 54Lb Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
5.	Professor Dan C. Boger, Code 54Bk Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
6.	Computer Technology Programs Code 37 Naval Postgraduate School Monterey, California 93940	1
7.	Navy Fleet Material Support Office P. O. Box 2010 Mechanicsburg, Pa. 17055	1
8.	Fleet Material Support Office P. O. Box 2010 Code 92 Mechanicsburg, Pa. 17055	1
9.	LCDR James L. Conroy Box 35 Brady, Nebraska 69123	2
10.	Lt. Michael T. Fuqua 705 Paseo Del Rey Chula Vista, California 92010	2
11.	LT. Julius J. Sisco 335 Hamilton Avenue Apt. 74 Norwich, Connecticut 06360	2

END

DATE
FILMED

5 -83

DTIC